12-2018

Implementation and Applications of Art-directable Ocean Simulation Tools

Jingcong Zhang
Clemson University, zhjingcong@gmail.com

Implementation and Applications of Art-directable Ocean Simulation Tools

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Fine Arts
Digital Production Arts

---

by
Jingcong Zhang
December 2018

Accepted by:
Dr. Jerry Tessendorf, Committee Chair
Dr. Eric Patterson
Dr. Victor Zordan

Abstract

Ocean effects are important aspects of the filmmaking. They help to establish emotions and dynamism via the behaviors of the oceans, and provides the different atmosphere for storytelling by creating various ocean scenarios. Gilligan is a prototype environmental scene simulator, whose core technique of the ocean simulation has been widely used in feature film productions. This thesis develops ocean simulation tools working with Maya and Houdini with the techniques provided by Gilligan. The Gilligan-Maya workflow executes ocean simulation methods in Gilligan to simulate oceans. The Gilligan-Houdini workflow integrates Gilligan into Houdini, containing a Houdini wrapper of Gilligan, along with a series of Houdini digital assets to support the usage. Artists can use these tools to generate ocean effects, with controls to simplify the production workflow, and well-exposed to all the simulation data for advanced development. This thesis demonstrates several applications with different ocean effects scenarios: ocean environment creation, ocean with floating objects, and ocean character effects.

Acknowledgments

I would like to express my sincere gratitude to my thesis advisor Dr. Jerry Tessendorf for his guidance and support during my Clemson DPA years. This thesis would not be done without his knowledge. The production of short animation *Bait* with Dr. Tessendorf inspired me to develop the ocean simulation tool with Gilligan. Thank Dr. Tessendorf for spending time with me on nearly every aspects of this thesis.

I would like to thank my committee member Dr. Eric Patterson for his advice on shading. I would like to thank my committee member Dr. Victor Zordan for his support and help.

I also want to thank Mr. Charles Trippe for his help on the Houdini issues. Finally, I would like to offer special thanks to my parents, friends, and Mr. Yifei Wu for their love and supports. I would like to thank Clemson DPA for providing such a nice environment for animation learning and productions.

Table of Contents

Title Page . . . . . **i**

Abstract . . . . . **ii**

Acknowledgments . . . . . **iii**

List of Tables . . . . . **v**

List of Figures . . . . . **vi**

1 Introduction **1**

2 Related Work **5**
2.1 Moana Water Character Scheme 5
2.2 Houdini Ocean Tools 7
2.3 Gilligan, Spectral Waves and eWave 9
2.4 Spectral Waves in Peanut Butter Jelly with Renderman 10

3 Implementation **12**
3.1 Gilligan-Maya Workflow 12
3.2 Gilligan-Houdini Workflow 15

4 Applications **30**
4.1 Ocean Environment Created by Gilligan-Maya Workflow 30
4.2 Ocean Environment Created by Gilligan-Houdini Workflow 31
4.3 Wakes of Floating Objects 33
4.4 Ocean Character 35

5 Conclusions and Discussion **39**

Appendices **40**
A Source Code of Houdini Wrapper 41
B Houdini Setup of Floating Animation Generator 50

Bibliography **53**

List of Tables

3.1 Houdini Gilligan Ocean custom SOPs and their usage 16
3.2 Houdini custom nodes used in each operations of the workflow shown in Figure 3.3 . 17

List of Figures

1.1 Ocean surfaces in *Waterworld(1995)*, Rhythm & Hues Studios and *Titanic(1997)* . . 2
1.2 Ocean Interaction in *Moana(2016)*, Disney Animation Studios [16] 2
1.3 Ocean character in *Moana(2016)*, Disney Animation Studios 3
1.4 A bottle containing a full-sized ocean (left) and water wall (right) in *Pirates of the Caribbean: Dead Men Tell No Tales(2017)*, MPC 3
1.5 *Bait(2017), Clemson DPA* 4

2.1 Water as character in *Moana(2016)*, Disney Animation Studios[15] 6
2.2 Small scale interactions and large scale walls of water in *Moana(2016)*, Disney Animation Studios [4] 7
2.3 Spiral Ocean by using wave instancing tool in Houdini, SideFX [14] 8
2.4 Ocean environmental scene generated via Gilligan, Gilligan demo 9
2.5 Ocean scene in *Peanut Butter Jelly(2014)*, Clemson DPA [6] 11
2.6 Ocean surface generated from the geometry, the displacement texture maps and RLOS [6] 11

3.1 Gilligan-Maya workflow 13
3.2 Ocean shader network in Gilligan-Maya workflow 14
3.3 Gilligan-Houdini workflow; the orange operations consists of Houdini custom SOPs, the other colored operations are developed as Houdini digital assets (refer to Table 3.2 and Table 3.1 for more details) 15
3.4 Houdini node network of wave surface simulation 18
3.5 Internal node network of Ocean Solver 19
3.6 Parameter interface of Gilligan Ocean Wavesurfer 20
3.7 Ocean surfaces in small patch size, large patch size and their merged layer 20
3.8 Visualization of ocean surface (left) and the corresponding rendered image (right) . . 21
3.9 Houdini node network of eWave simulation 22
3.10 Surface shader in Ocean HeightMap shader 22
3.11 Mantra texture baking, SideFX[12] 23
3.12 Basic eWave setup 24
3.13 Parameter interface of Gilligan Ocean eWave 24
3.14 Node network of bounding box clipping method for eWave simulation 25
3.15 Top view of horizontally distorted ocean surface with eWave simulation patch 27
3.16 eWave combination without (left) and with (right) remapping 27
3.17 Top view of horizontally distorted ocean surface with eWave simulation patch 28
3.18 Ocean displacement shader 28

4.1 Ocean environment created by the ocean simulation following Gilligan-Maya workflow in *Bait(2017)*, Clemson DPA 31
4.2 Reference sea environment images shot at Tomales point, CA 31
4.3 Nuke tree to create ocean fog effects 32

4.4 CG sea environment created by Gilligan-Houdini 32
4.5 An animated floating object (blue object) with proxy geometry (grey triangle) and rough ocean surface 33
4.6 eWaves applied onto the ocean surface; the ocean surface without eWaves (left), the ocean surface with eWaves marked as eWave displacement value (middle), the ocean surface with eWaves (right) 34
4.7 Ocean scene with floating objects moved by animation generator; the ocean surface without eWaves (left), the ocean surface with eWaves marked as eWave displacement value (middle), the ocean surface with eWaves (right) 34
4.8 Guided path curves of floating objects 35
4.9 Composited image of the ocean scene with floating objects in Figure 4.7 35
4.10 Concept materials of the normal status and the ocean status 36
4.11 Concept images of the transition 36
4.12 An example of transition mask 37
4.13 Three frames of the ocean character transition: the start of the transition, the middle during the transition and the final result of the transition 38

1 Houdini setup of floating animation generator 51
2 Houdini setup of transformation calculation of projected proxy geometry onto ocean surface 52
3 Houdini setup example of transformation calculation VOP 52

Chapter 1

Introduction

Ocean environment is an important and difficult location for filmmaking, consequently, ocean simulation is a topic of interest in the visual effects industry. The ocean has a highly dynamic behaviour, ranging from a quiet sea to an agitated storm, from small turbulent waves to enormous shorebreaks. Darles et al. present a survey of the techniques for simulating oceans [3]. Creating realistic, aesthetically pleasing ocean simulations is particularly formidable, when it must interact with a character. CG oceans are a useful tool when they can be controlled in an art-directable fashion.

In practice, an ocean simulation system for visual effects needs to be art-directable, and easily controlled to match artists' concept designs and reference oceans. Houdini Ocean Toolkit [13] and Bifröst Ocean Simulation System (BOSS) [2] are widely used in visual effects studios. Many visual effects studios implement their own simulation tools, for example, the simulation system described in [7]. For ocean surface simulation, it is common to create ocean wave height fields based on spectral models of ocean spatial statistics. The spectral models are widely used to produce beautiful ocean surfaces [8]. They use spatial sine waves to describe the surface, computed by an empirical wave spectrum and a Gaussian pseudo-random number generator [18]. Early applications of this approach include the famous CG water scenes in the films *Waterworld(1995)* and *Titanic(1997)* (Figure 1.1). Considering the challenges of spectral models in practice, like the situations of shallow water or directional spreading, Horvath advocated the TMA spectrum as a visually pleasing alternative [8]. Inspired by these techniques, Disney Animation Studios and Moving Picture Company (MPC) created their custom ocean toolkits, which have been used in the production of *Moana(2016)*

Figure 1.1: Ocean surfaces in *Waterworld(1995)*, Rhythm & Hues Studios and *Titanic(1997)*

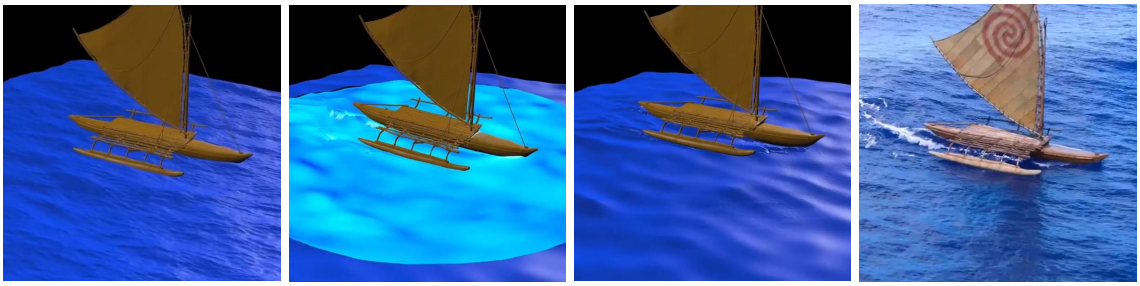

Figure 1.2: Ocean Interaction in *Moana(2016)*, Disney Animation Studios [16]

and *Pirates of the Caribbean: Dead Men Tell No Tales(2017)* [5] [7] [11]. For ocean interaction with characters or objects, it is common to create a separate simulation near the characters or objects, and embed it in the middle of the ocean. Disney used a custom blend node to combine separate particle simulations with ocean height fields in *Moana(2016)* [11] (Figure 1.2). Houdini's Ocean Flat Tank tools can run FLIP tanks to simulate the entire surface after initializing from the ocean surfaces. This kind of approach was also used in *Pirates of the Caribbean: Dead Men Tell No Tales(2017)* by MPC with the use of Bifröst [7]. Meanwhile, Tessendorf provides an approach to simulate interactive water surface, named as "eWave" (derived from "iWave" in [18]) to create height fields, which generates simulations of ripples or wakes and is easily integrated with the ocean surface.

In production, the requirements for ocean simulation are typically to mimic reality and to create non-realistic ocean effects, or magic ocean effects, which advance the story's background and express emotions. Special CG ocean workflows and systems have been developed to deal with this kind of situations. Taking the example of *Moana*, Disney treats ocean waves as a character. The ocean has a mind of its own and can form its water to grab objects, mimicking human traits such as head-nodding or high-fiving, as a way to communicate with Moana and her accomplices [1]

Figure 1.3: Ocean character in *Moana(2016)*, Disney Animation Studios

Figure 1.4: A bottle containing a full-sized ocean (left) and water wall (right) in *Pirates of the Caribbean: Dead Men Tell No Tales(2017)*, MPC

(Figure 1.3). This builds into additional behaviours, narratives and emotional states desired for performing water development. In order to meet the requirements of behaviours and emotions, Disney developed a workflow with easily controlled simulation guides, with animation and effects departments working in collaboration to determine water character's performance [4].

The ocean effects in *Pirates of the Caribbean: Dead Men Tell No Tales* are another examples of this. MPC faced creative challenges to produce an ocean in a bottle, and the waterwall effects with the entire ocean spliting into two, and then collapsing together (Figure 1.4). MPC adapted workflows mixing Houdini and Bifröst, with guided geometry and animation for the simulation [7].

The workflows to create the ocean environment and the effects of ocean on shapes are varied. A workflow of Gilligan [17] with Maya and Arnold has been developed to produce the ocean scene in the short animation *Bait(2017)* (Figure 1.5). This workflow used the spectral model and eWave to simulate the ocean by using the methods provided by Gilligan. Meanwhile, the effects of ocean on shapes are usually undertaken by the fluid simulation with careful animation, while Disney bound the ocean simulation onto the wall shape. This motivated a technique to warp the ocean surface with the desired shape. Oceans confined to the surface of the non-planar geometry

Figure 1.5: *Bait(2017), Clemson DPA*

would be a valuable feature in the magic ocean effects. An ocean simulation tool based on the Gilligan workflow used in *Bait* is re-implemented with these new features. It is useful for several ocean scenarios. For example, it can be used to create a photo-realistic ocean environment; it can easily generate the ocean background with floating objects; and it helps to create oceans on shapes using the ocean surface warping technique. It is a Houdini wrapper for spectral waves and "eWave" described previously, along with a series of Houdini digital assets tools and custom shaders.

Chapter 2

Related Work

Animation studios and visual effects companies have their own ocean effects workflows. Special workflows have been developed for particular ocean effects. The ocean effects in *Moana* are the one of the most famous examples. It uses several techniques for different ocean scenes, as described in section 2.1. Houdini, as one of the most widely used FX softwares, offers a tool set to create ocean related effects, ranging from generating standard ocean environment to creating ocean on shapes. These techniques and workflows provide insights of how to generate a standard ocean simulation.

Gilligan, equipped with one of the most widely used ocean simulation technique, spectral waves, is a useful tool to build a custom ocean simulation system [17]. This technique has been directly used in short animation production *Peanut Butter Jelly(2014)*. Gundersen created a Renderman displacement shader, Renderman Layered Ocean Shader (RLOS), which makes it easier and more efficient to generate realistic ocean, based on the previous research in *Peanut Butter Jelly* [6].

2.1 Moana Water Character Scheme

Disney's *Moana(2016)* was set in an environment inspired by the Pacific Islands, which made the ocean a prominent setting throughout the film. Disney had two types of water for their ocean generation: procedural water and simulated water[5]. How *Moana* performs ocean water is a great inspiration. The ocean have its own life. There were two different irregular base shapes of ocean water in *Moana*: water as character and water wall. Different shapes and scales of the ocean

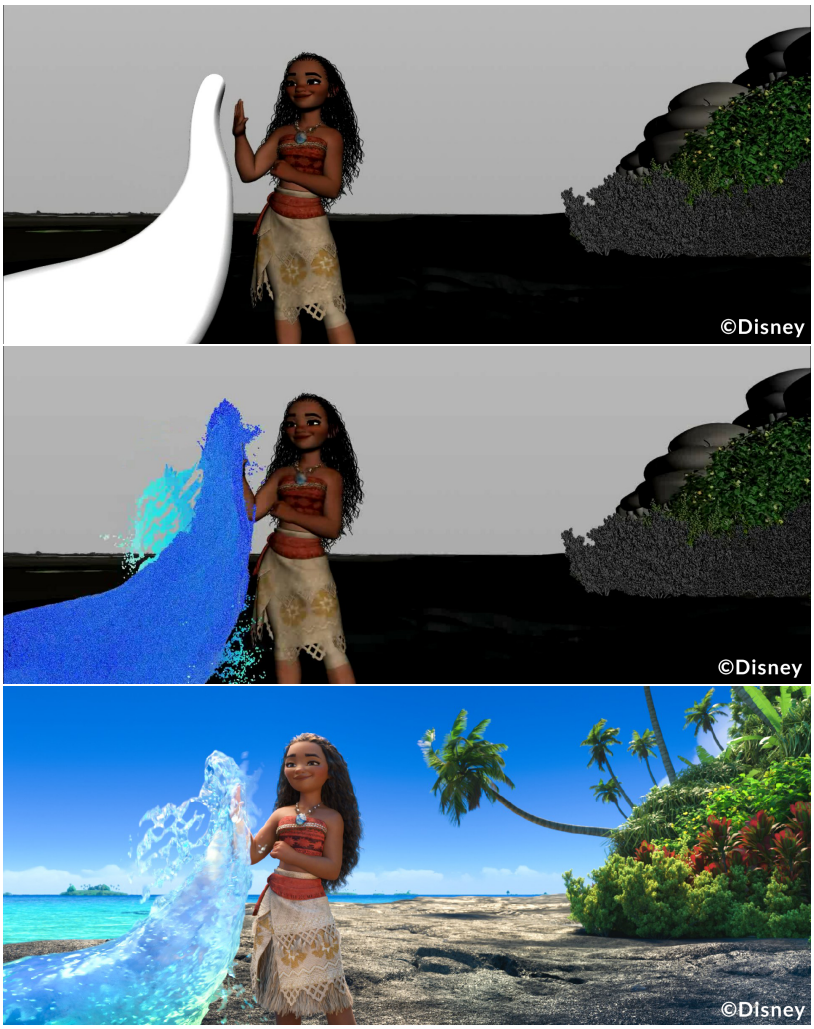

Figure 2.1: Water as character in *Moana(2016)*, Disney Animation Studios[15]

were used to express different emotions dynamically.

The character water, which played with toddler Moana, and helped Moana all the way along her trip, required emotion and dynamism for its establishment. Disney used simulated water based on the character animation. They used wire simulation to provide secondary simulation and fluidity of motion, and then created a guided fluid simulation [4]. This method was usually used in calmer moments. Another solution for more active scenes was to run a fluid simulation in the rest position, then bind it to the animation, and add a secondary simulation for interactive narratives and splashes, which can add a physically realistic component to the character. Moana high-fiving with water is a good example of this solution (Figure 2.1) [15].

Figure 2.2: Small scale interactions and large scale walls of water in *Moana(2016)*, Disney Animation Studios [4]

The water walls featured parting of the ocean with completely different ocean scale for each sequence (Figure 2.2). The shoreline parts for toddler Moana, which needed to be quiet, non-threatening ocean, were created by the procedural water techniques. Disney modeled the base shapes of the wall, and then applied the ocean deformations on the top portions of the walls. But the opening of the ocean in the climactic sequence, where the water of ocean sustained with highly dynamics, encompassed several techniques. It used the art-directed flow curves and created reusable high resolution wall panel simulations which were deformed and replaced by the modeled wall designs [4].

2.2 Houdini Ocean Tools

Houdini offers several solvers related to ocean simulation. The ocean shelf tools in Houdini 16 includes the ocean surface generator in different scales, the guided ocean layer, several FLIP tanks dealing with ocean and the whitewater generator. Those provides clues how effects artists will use Houdini to create ocean simulation. For example, users can create the ocean surface by using the ocean spectrum and ocean evaluation tools. Users can also create dynamics ocean effects with a lot of foam and interaction with objects by using the FLIP solver, displace the flat ocean from the one previously created, and mask the interaction with the velocity.

Houdini's Ocean tools have been significantly redesigned for Houdini 16, offering more control over the look and timing of oceans, render-time evaluation of ocean spectra, and improved tools for integration with FLIP simulations. Houdini utilizes Ocean Spectrum SOP to create data, which can be evaluated via Ocean Evaluation SOP or Ocean Sample Layers VOP. In Ocean Spectrum SOP, Houdini offers ocean spectrum, which is a measure of the amplitudes of a bunch of waves of different frequencies. Houdini procedurally generates a set of frequencies that correspond to an ocean of a

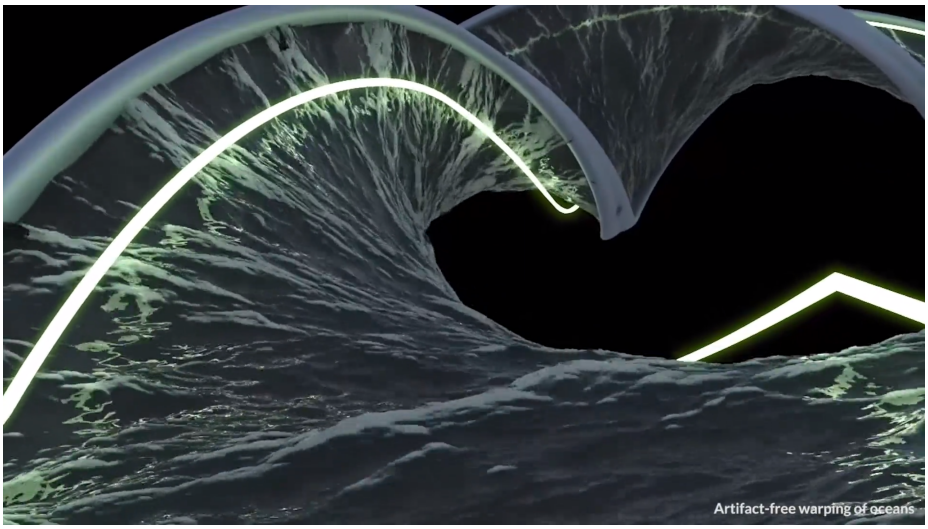

Figure 2.3: Spiral Ocean by using wave instancing tool in Houdini, SideFX [14]

particular depth with wind moving in a particular directions with some other artist-controlled parameters. And then it generates a set of amplitudes for these waves by taking all of those data using FFTs as actual displacement which is applied to a grid in world space. This process is represented with three 2-D volumes for amplitude, phase and angular frequency of the waves. Houdini 16 has a layered architecture for oceans. Ocean spectra and spectral geometry[1] were used to define the ocean, arbitrary ocean spectra can be merged together to generate a combined spectra. In Ocean Evaluation SOP or Ocean Sample Layers VOP, Houdini evaluates the ocean data outputs, which are displacement, velocity and cusp. This process is usually undertaken at the render time [13].

Houdini 16 offers a technique for creating ocean on shapes. It makes ocean waves be applied artifact-free even to deformed or warped base grids. This technique is called wave instancing, it instances patches of spectral ocean onto points based on their various point attributes and overall instancing parameters. Both 2D and 3D point clouds are supported [14]. Those point attributes obey the typical copy instance attributes of Houdini to orient the patches of ocean, to scale up their size and generally give a lot of control. For example, when making the ocean waves move along a surface, spectral ocean patches can be instanced onto points scattered on the surface, with attribute "N" defining the direction of wave's movement, and attribute "up" orienting the wave displacement. The instancing parameters provide overall controls on the instancing as well as randomize them. With wave instancing, deformed ocean effects can be easily created based on user-defined point

[1]The structure is used to store the spectra data in Houdini.

Figure 2.4: Ocean environmental scene generated via Gilligan, Gilligan demo

clouds. For example, the spiral ocean in Figure 2.3 with the proper rotation of velocity and cusp, was generated by instancing spectral ocean patches onto 3D point cloud.

2.3 Gilligan, Spectral Waves and eWave

Gilligan is a prototype environmental scene simulator, developed by Dr. Tessendorf and graduated students, for simulating and rendering scenes containing ocean surfaces, interaction of water surfaces with objects on the water, clouds, and atmospheres [17]. Figure 2.4 is an example of ocean environmental scene generated from Gilligan. Gilligan implements spectral waves and eWave for the water surfaces and interaction of water surfaces. Some components of Gilligan are written in C++, with Python interfaces automatically generated by SWIG[2].

2.3.1 Spectral Waves

Gilligan utilizes spectral waves to generate ocean surface simulation. Spectral waves combine a physical model and a statistical model. The physical model describes the relationship between

[2]SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages. It is used with different types of target languages including common scripting languages such as Javascript, Perl, PHP, Python, Tcl and Ruby.

different frequencies and the magnitude of their corresponding wavevectors, the horizontal vector that points in the direction of travel of the wave, within an ocean of a certain depth. For example, in deep ocean, the large low frequency rolling waves move much faster than the small choppy waves [13]. This relationship is called dispersion, which is a function of frequency driven by wavelength of wave and other physical properties like bottom depth and surface tension. There are several dispersion relations, such as for deep water or shallow water. A combined dispersion relation is used in Gilligan [17]. The statistical model for ocean spectra have been compiled by Massel [9]. In the statistical model, the wave height is considered a random variable of horizontal position and time, and also based on the decomposition of the wave height field as sine and cosine waves, which are represented in the fourier domain and estimated by the ocean spectrum [18]. Gilligan offers option of selecting from many combinations of frequency spectrum and directional spectrum. Finally, a random ocean wave height field can be generated by combining Gaussian random numbers with the spatial spectrum. This process can be efficiently done in the Fourier domain.

2.3.2 eWave

Gilligan utilizes eWave for interaction simulation of wave surfaces. eWave is the exponential solution for iWave problem in [18] with a replaced propagation algorithm based on FFTs. eWave is mathematically identical to that of the spectral model solution, but arranged in a way that supports interaction with objects [17]. eWave uses a 2D map of the interaction of obstructions with a value of 1 where there is no intersection and 0 where there is intersection, called an "obstruction map". This can also help to identify the sources that drive surface disturbances. The simulated eWave layer can be applied along with spectral ocean surfaces, as a single displaced ocean surfaces. eWave simulation is a relative new technique to create the interaction with the ocean.

2.4 Spectral Waves in Peanut Butter Jelly with Renderman

The ocean surface in short animation *Peanut Butter Jelly(2014)* was created by the technique of spectral waves (Figure 2.5). Based on the research of ocean creation in *Peanut Butter Jelly*, Gundersen developed a Renderman displacement shader, RLOS, to generate the ocean spectra at the render-time [6]. He compared three usages of spectral waves as Figure 2.6 shows: directly rendering the ocean surface geometry generated via Gilligan, rendering ocean surface using the displacement

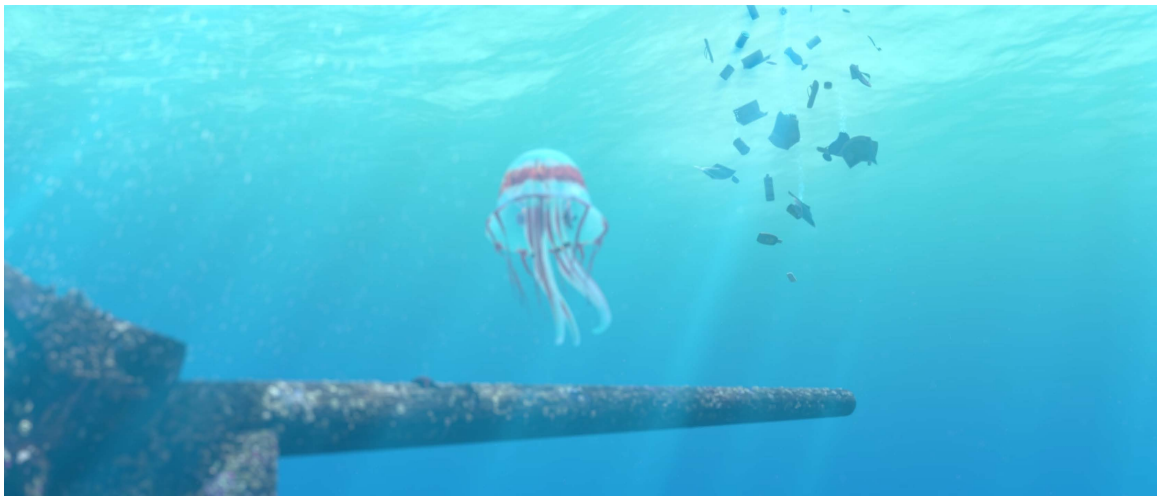

Figure 2.5: Ocean scene in *Peanut Butter Jelly(2014)*, Clemson DPA [6]

Figure 2.6: Ocean surface generated from the geometry, the displacement texture maps and RLOS [6]

textures, and generating ocean spectra at the render-time. The ocean surface generated by RLOS presented more details compared with geometry mesh and texture maps. The way RLOS to generate ocean surface offered a more efficient way. It was a trade-off to use high-defined geometry mesh or texture maps in an efficient way, because the geometry mesh was too heavy with all the height details to render, while the texture maps may cause pixelation artifacts without enough texture resolutions. Gundersen developed RLOS by using RSL plug-in with pre-existing C++ code in Renderman. RLOS generated the height information at render time by calling the spectral waves algorithm [6].

Chapter 3

Implementation

This thesis develops ocean simulation tools with workflows of Gilligan and the DCC (digital-content-creation) packages, aiming to produce ocean environment and the effects of the ocean on shapes. Two workflows are implemented: Gilligan-Maya workflow and Gilligan-Houdini workflow. The Gilligan-Maya workflow is designed to create the ocean environment in *Bait*. It executes the ocean simulation methods to generate required simulation data. For example, the ocean surface mesh and the displacement map textures. It uses a custom ocean shader in Maya and Arnold to generate the final look. However, the Gilligan-Maya workflow is complicated in some cases. Therefore, the ocean simulation tool is reinforced in Houdini with new features such as binding the ocean simulation onto a non-planar geometry, automatically generating floating animations for secondary objects, and etc. The Gilligan-Houdini workflow contains a Houdini wrapper of Gilligan to simulate the ocean, and a series of Houdini digital assets to support the simulations. The Houdini wrapper integrates Gilligan by using the Python APIs from Gilligan for the ocean surface simulation and eWave simulation. The Python code is shown in appendix A.

3.1 Gilligan-Maya Workflow

In order to create the ocean environment with several objects interacting with the ocean, three components needs to be considered: the ocean waves, the movements of the objects along with the ocean waves, and the wakes or ripples generated from the objects. The workflow for Gilligan with Maya is shown in Figure 3.1, with these three components marked as green, yellow

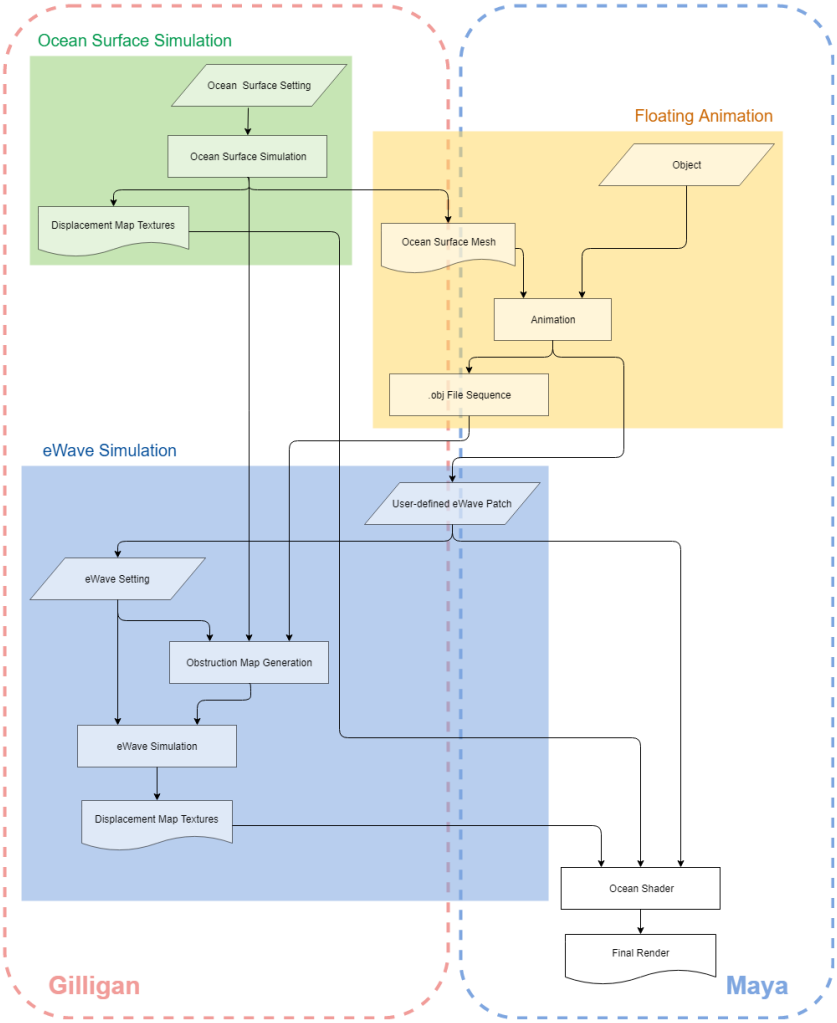

Figure 3.1: Gilligan-Maya workflow

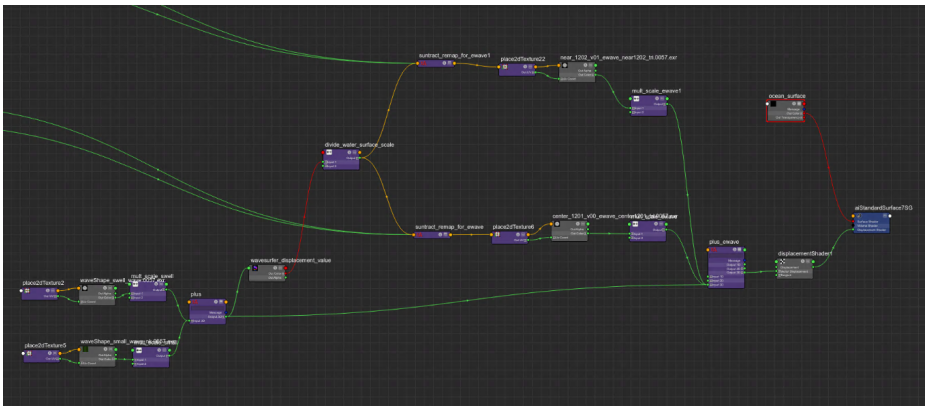

Figure 3.2: Ocean shader network in Gilligan-Maya workflow

and blue. The red box contains operations executed in Gilligan, while the blue box contains all the operations undertaken in Maya. An ocean surface is simulated in Gilligan, exporting several layers of displacement map textures and a low-resolution mesh of the ocean surface in .obj format using Gilligan's internal methods. This mesh can be used as a reference geometry for animation. In order to create interaction effects, the artist animates objects based on the reference geometry, exports the animated objects as a sequence of .obj files, and specifies the patch areas for the eWave simulations in Maya. Then the eWaves are simulated in Gilligan using the eWave patch areas and the animations. Finally, in Maya, a custom ocean shader network using Arnold aiStandardSurface shader with vector displacement is applied to a plane to create the final ocean look, which combines displacement map textures of the ocean surface and the eWave together. Figure 3.2 shows the ocean shader network in Hypershade in Maya.

As it shows in Figure 3.1, this workflow has Gilligan and Maya interleaved a lot. A lot of data has to be passed between Gilligan and Maya, such as the ocean surface proxy geometry from Gilligan to Maya, and animated object based on the ocean surface geometry from Maya back to Gilligan. It is because a lot of operations are executed from Gilligan's pre-existing methods; for example, the animation has to be exported from Maya as .obj files because Gilligan takes .obj mesh to generate the "obstruction maps" at each frame to simulate eWave. Gilligan is designed to create ocean environments. Therefore, effects like ocean on shapes cannot be perfectly handled with its pre-existing methods. A more efficient, user-friendly workflow based on Gilligan can be further developed. Houdini has the capability to wrangle a large amount of data, allowing internal simulation data easily to be stored and accessed. It provides Python APIs to modify internal geometry data,

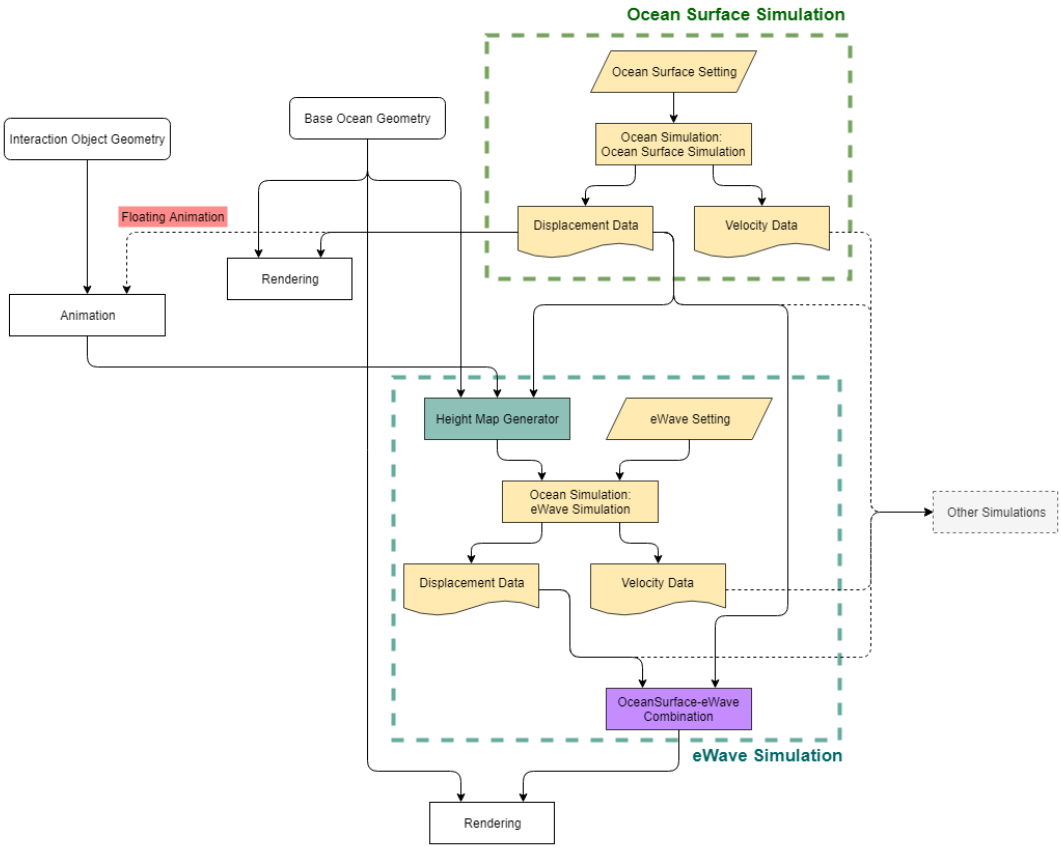

Figure 3.3: Gilligan-Houdini workflow; the orange operations consists of Houdini custom SOPs, the other colored operations are developed as Houdini digital assets (refer to Table 3.2 and Table 3.1 for more details)

making it easy to glue with Gilligan. The data of the geometry, other simulations, the custom fields, and the custom shaders share the same structure, and can be easily accessed and modified serializablly, providing more flexibility to the development. It is worthwhile to develop an ocean simulation workflow of Gilligan with Houdini.

3.2 Gilligan-Houdini Workflow

The Gilligan-Houdini workflow is from the simulation stage using Gilligan's Houdini wrappers to the final rendering stage in Mantra, as Figure 3.3 shows. Artists can define any uv-unwrapped geometry as a base geometry to hold ocean wave data to generate the ocean look. They can also specify the interaction objects to generate interactions on the ocean. The simulation stage provides

Gilligan Ocean Custom SOP Name	*Usage*
Gilligan Ocean Setup	Constructing two 2D vector volumes for displacement data and velocity data
Gilligan Ocean Wavesurfer	Setting up ocean surface simulation parameters of a single layer, such as typical height, direction, depth, etc.
Gilligan Ocean Merge	Merging several layers of ocean surfaces together
Gilligan Ocean eWave	Setting up eWave simulation parameters, such as capillary, displacement scale, etc.
Gilligan Ocean eWave Source	Updating height source (the obstruction map data) for the eWave simulation
Gilligan Ocean Simulation	Updating ocean simulation data

Table 3.1: Houdini Gilligan Ocean custom SOPs and their usage

simulations for the ocean surface and the eWave, which is performed in Houdini geometry node (SOP) defined by Python. Two 2D Houdini vector volumes holds the output data of each ocean layer, as the displacement data and the velocity data. Gilligan's Houdini wrapper initializes and updates simulation data within these vector volumes. For eWave simulation, the obstruction maps are automatically generated after the ocean surface simulation, fed into eWave network as height source. A custom Houdini VOP node merges the simulations of the ocean surface and the eWave together. During the rendering stage, the ocean displacement is applied to the geometry based on its uv coordinates. The velocity data from the ocean simulations can also be used for other simulations, such as the initialization of the FLIP simulation.

The core part of the Houdini wrapper is the utilization of Gilligan's Python APIs to initialize and update Houdini volume data, which is defined as geometry data in Houdini. Therefore, the entire process should be performed in the Houdini geometry node (SOP) network in Geo objects. Among all the Houdini's SOPs, Python SOP is designed to modify input geometry using Python script. This is perfect for implementing Gilligan with Houdini in this system. Houdini offers an API called Houdini Object Model (HOM) to let developers get information from and control Houdini. Houdini can define the reusable geometry node (SOP) using Python. Several SOPs are defined using Python for Gilligan: "Gilligan Ocean Setup", "Gilligan Ocean Wavesurfer", "Gilligan Ocean Merge", "Gilligan Ocean eWave", "Gilligan Ocean eWave Source" and "Gilligan Ocean Simulation". Table 3.1 presents the usage of these custom SOPs. The first four nodes are used for simulation initialization, and will run only once after setup. The rest nodes are used for updating, and will execute at each frame or sub-frame.

Operations in Figure 3.3	*Houdini Custom Nodes*
Ocean Surface Setting	Gilligan Ocean Setup SOP Gilligan Ocean Wavesurfer SOP
Ocean Simulation	Ocean Solver SOP Gilligan Ocean Simulation SOP
eWave Setting	Gilligan Ocean Setup SOP Gilligan Ocean eWave SOP Gilligan Ocean eWave Source SOP
Height Map Generator	eWave Height Map digital asset
OceanSurface-eWave Combination	eWave Combination VOP

Table 3.2: Houdini custom nodes used in each operations of the workflow shown in Figure 3.3

Several other nodes are also developed to support Gilligan Houdini wrapper. "Ocean Solver", a Houdini Solver geometry node, holds the "Gilligan Ocean Simulation" node for simulation, and provides updating scheme at each sub-frame. "eWave Combination", a Volume VOP node, combines the ocean surface layer and the eWave layer. A digital asset, called "eWave Height Map", consists of a SOP network containing several SOPs and Volume VOP nodes. It is used for processing eWave height map. Meanwhile, custom shaders are developed for generating obstruction maps, differentiating underwater parts for objects, final rendering and other usages.

All the operations in the workflow consists of these nodes, as it shows in Table 3.2. The detailed implementation is described in the following sections.

3.2.1 Wave Surface Simulation in Houdini

Figure 3.4 shows an example of two layers of ocean surface simulation. With ocean simulation data set up, two layers of ocean surface simulations defined and merged together, the ocean simulation data updates via "Ocean Solver".

"Ocean Solver" is the core node in the ocean surface simulation network, which updates the ocean simulation data. It is a Houdini Solver geometry node, which allows running a SOP network iteratively over the input geometry, with the output of the network from the previous frame serving as the input for the network at the current frame. Users can also define the sub-steps via this node, creating a more precise simulation. The "Ocean Solver" takes three inputs: ocean setup, eWave setup and ocean surface setup. Its ocean setup input serves as the input geometry, which receives displacement and velocity volumes defined by the "Gilligan Ocean Setup" node. Figure 3.5 shows the SOP network inside the "Ocean Solver" node. It updates the input geometry, and outputs the

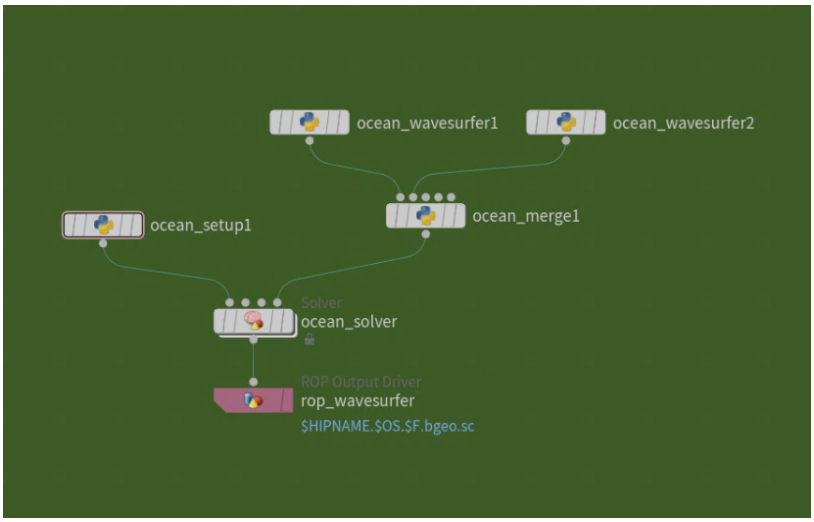

Figure 3.4: Houdini node network of wave surface simulation

updated displacement and velocity volume data. The simulation uses the time-step calculated from the current frame rate and sub-steps via the "Ocean Solver" node. "Ocean Solver" only uses the ocean setup input and the ocean surface setup input for the ocean surface simulation.

"Gilligan Ocean Setup" node takes the user-defined ocean size, ocean lower-left corner (LLC) position, and ocean resolution as parameters to initialize two 2D vector volumes for displacement and velocity data. The actual world space scale and position of the ocean are defined by the ocean size along with the LLC position. The vector volumes are initialized with the bounding box calculated by the ocean size and LLC position. Their resolution in X and Z dimensions is defined by the ocean resolution, which is the simulation resolution. The volume data is remapped into the uv-space of base geometry.

"Gilligan Ocean Wavesurfer" and "Gilligan Ocean Merge" can both be used as ocean surface setup input of "Ocean Solver". Both of them generate the ocean WaveSurfer object, which is a factory class used to create different types of ocean surfaces in Gilligan. "Gilligan Ocean Wavesurfer" constructs ocean surface simulation via several parameters. The parameter interface is shown in Figure 3.6. Different types of ocean surfaces are suitable for different situations; for example, the deep ocean wave is suitable for desired rounded waves corresponding to a clear day, while the TMA wave is a shallow wave using TMA spectrum. "Gilligan Ocean Merge" generates a new WaveSurfer object by taking several layers of WaveSurfer object as inputs. "Gilligan Ocean Wavesurfer" allows

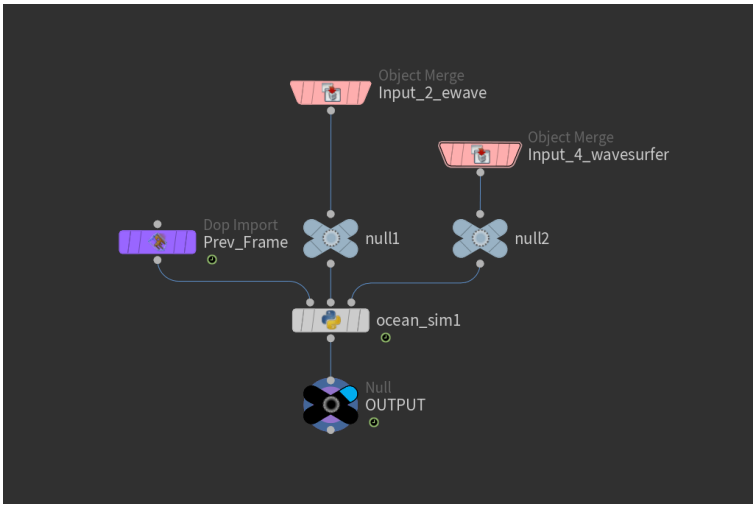

Figure 3.5: Internal node network of Ocean Solver

users to define the patch size and the resolution of each patch. Gilligan simulates each layer of ocean surface in particular patch size to generate repeatable displacement pattern along the ocean. Multiple layers of ocean surfaces in different patch size can be merged together to generate nice wave surface without hard repeating pattern. Figure 3.7 shows an example of the displacement data generated from two ocean surface layers in different patch size and their merged layer. Besides using "Gilligan Ocean Merge" node to combine multiple layers of ocean surfaces, the combination can also be performed during post-simulation stage or at the render-time by simply summing up each layer's displacement data of each voxel at the same position. This helps to add details to the main simulation without re-simulating.

Meanwhile, it is noteworthy how the node chain passes through Gilligan object during the entire simulation. The WaveSurfer object from Gilligan is considered as custom data field, beyond what is already stored by Houdini, which is arbitrary data on individual node. "Gilligan Ocean Wavesurfer" and "Gilligan Ocean Merge" use methods offered by HOM to store this kind of per-node user-defined data, making the corresponding WaveSurfer object associated with themselves, and capable to be retrieve by other nodes such as the "Ocean Solver" node. However, the cached data on each node is not serializable, which means the WaveSurfer object cannot be directly passed through the node chain to the bottom nodes. In order to retrieve the WaveSurfer object data in "Ocean Solver", a new detail attribute called "wavesurferpath" in "Gilligan Ocean Wavesurfer" and "Gilligan Ocean Merge" stores the path of current node. In this way, the WaveSurfer objects can be

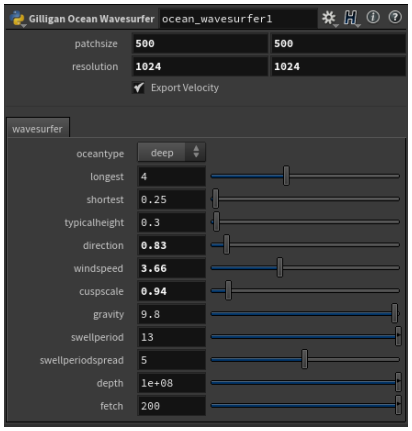

Figure 3.6: Parameter interface of Gilligan Ocean Wavesurfer

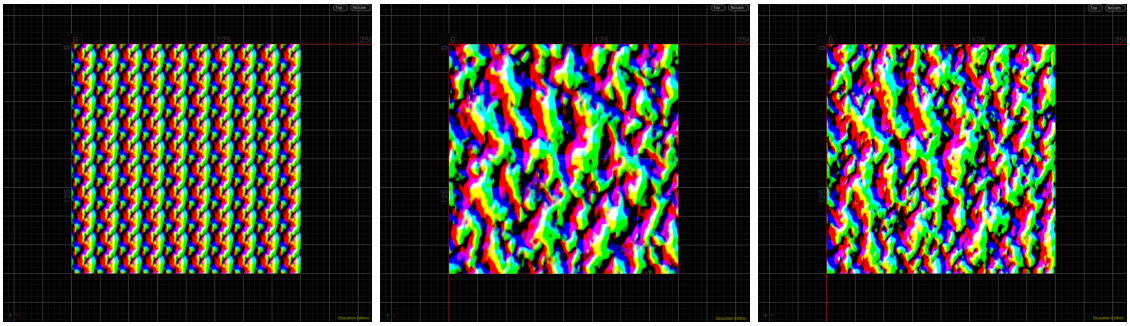

Figure 3.7: Ocean surfaces in small patch size, large patch size and their merged layer

accessed by any node in the same geometry network by retrieving the data from the corresponding node path stored in its detail attributes.

With volumes and ocean surface object setting up, "Gilligan Ocean Simulation" executes the Gilligan ocean simulation by calling update method for the WaveSurfer object, and retrieves the simulation data of each volume voxel based on its world space position at current frame or sub-frame.

In order to make the tool of ocean surface generation follow the rule of "What you see is what you get", a visualization tool implemented by a Houdini Attribute VOP node presents a rough displaced geometry mesh for the ocean surface (Figure 3.8).

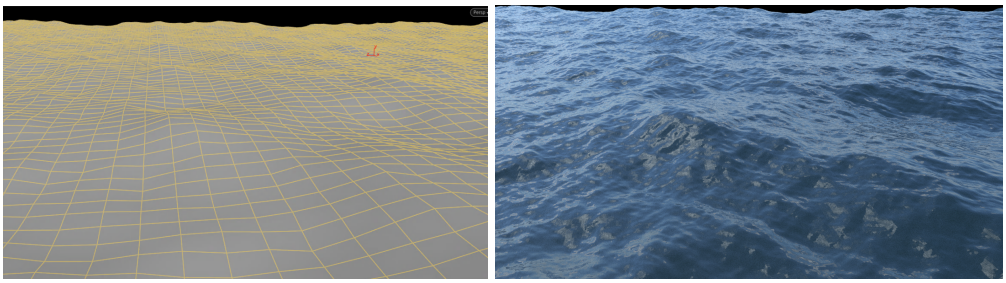

Figure 3.8: Visualization of ocean surface (left) and the corresponding rendered image (right)

3.2.2 eWave Simulation in Houdini

3.2.2.1 eWave Setup and Simulation

Figure 3.9 shows an example of eWave simulation. The "Ocean Solver" uses the ocean setup input and the eWave setup input instead of the wave surface input. As the same as wave surface simulation, the ocean setup uses "Gilligan Ocean Setup" node to create volume data in particular position, scale and resolution, and the eWave setup generates Gilligan eWave object for simulation. The eWave simulation process is similar with the ocean surface simulation. The data retrieval follows the same process as ocean surface simulation: updating eWave object and retrieving simulation data for each volume voxel using its world space position. Meanwhile, eWave object in Gilligan offers the same APIs as WaveSurfer object to update and get simulation data.

However, eWave object generation is a different process from the WaveSurfer object generation. It happens after ocean surface simulation, and can be separated as two parts: eWave height map generation and eWave setup. eWave height map serves as the "obstruction map" for eWave simulation, which is a 2D map to identify the interaction of obstructions and ocean geometry with the value of 1 for where there is interaction and 0 for the area without interaction. The ocean geometry is displaced based on the remapped ocean surface simulation data. Bake Texture ROP is used to generate eWave height map, because it has the capability to map 3D space information generated from shader onto uv-space. "Ocean HeightMap" custom shader serves to generate such information. It is attached to the ocean geometry, containing a surface shader to identify interaction (Figure 3.10) and an ocean surface displacement shader. The surface shader runs after displacement shader, and performs an intersection test with obstruction geometry. The intersection test algorithm is simple: shooting rays from ocean geometry along its normal direction, testing the ray intersections with the

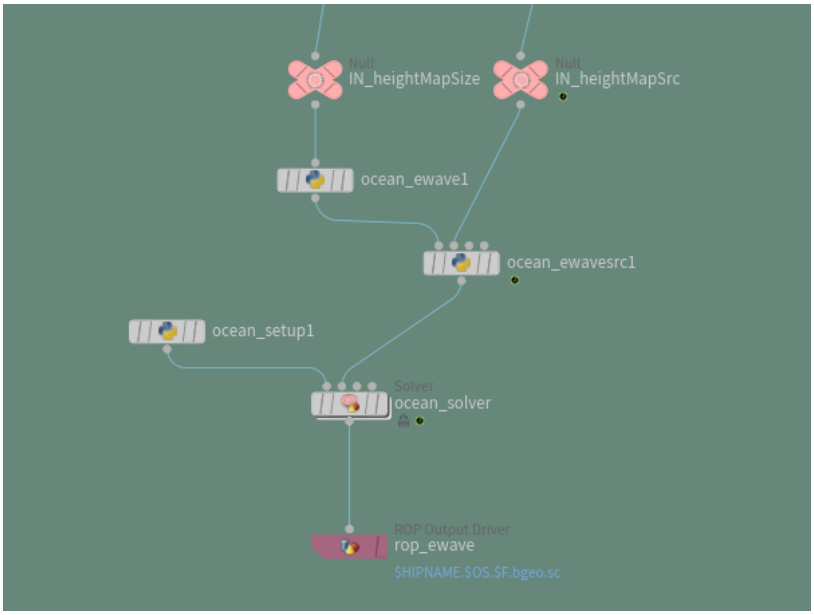

Figure 3.9: Houdini node network of eWave simulation

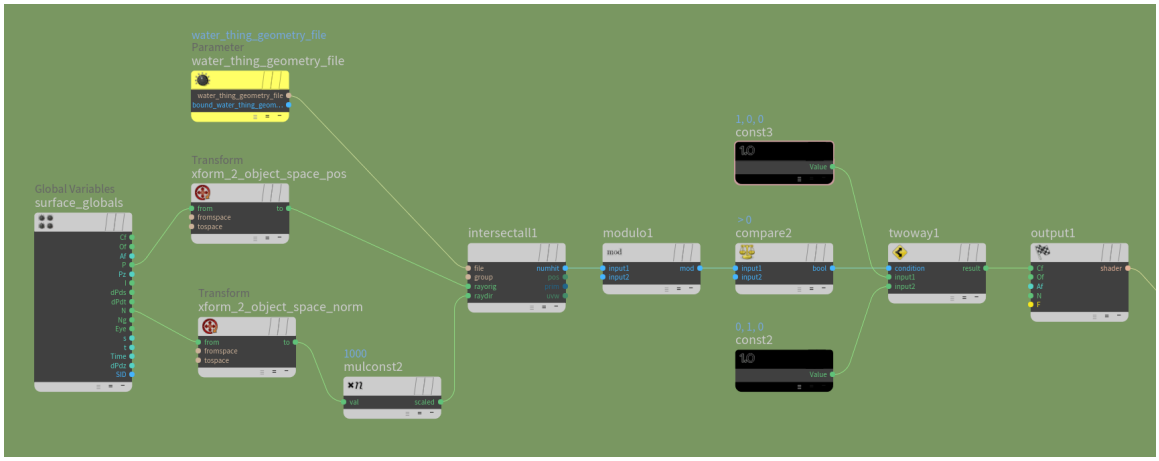

Figure 3.10: Surface shader in Ocean HeightMap shader

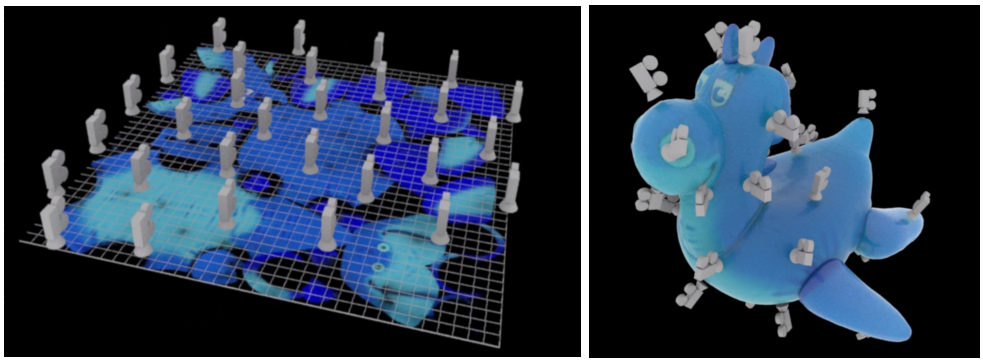

Figure 3.11: Mantra texture baking, SideFX[12]

obstruction geometry along the ray path, setting intersection status to true if there is an odd number of intersections, and false if there is an even number of intersections (Equation 3.1). The intersection test uses Houdini Intersect All VOP node, which computes all the intersections of the specified ray with the geometry, with the position of ocean geometry as the ray origin, and the normal of ocean geometry as the ray direction. The position and normal attributes read from global variables have to be transformed into object transform space, since they are in camera transform space by default in shader network. With the user-specified shader, which is the "Ocean HeightMap" custom shader in this case, Bake Texture ROP renders the texture image by moving camera over pixels, sending rays to the uv image and storing the computed color in each pixel. Algorithm 1 and Figure 3.11 present how Bake Texture ROP works in Houdini Mantra[12]. In this way, intersection information is mapped as texture, creating an eWave height map.

$$intersection = \begin{cases} false & \text{if number of intersections} \mod 2 = 0 \\ true & \text{otherwise} \end{cases} \tag{3.1}$$

Figure 3.12 shows how a basic eWave object is set up. This happens after eWave height map generation. A 2D Houdini volume is used to hold the eWave height map data. The Volume VOP node named "Gilligan Ocean Setup" maps the eWave height map texture onto volume grid, which mainly uses Houdini Color Map node. Similar with the "Gilligan Ocean Setup" node, which defines a patch of ocean in particular resolution, "Gilligan Ocean eWave" uses patch size, resolution and LLC to define a patch of eWave area. It constructs Gilligan eWave obejct with defined eWave patch and several other user-controlled parameters. Its full parameter interface is showed in Figure 3.13.

Algorithm 1 Mantra texture baking algorithm in Bake Texture ROP[12]

```
1: function BakingTexture
2:     for each pixel do
3:         for each sample do
4:             Compute camera ray
5:             Send ray into scene
6:             Store color in sample
7:         end for
8:         Filter samples into a pixel
9:         Store pixel in image
10:    end for
11: end function
```

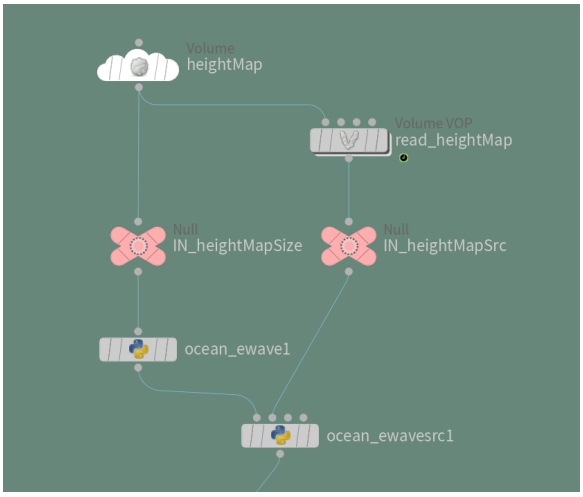

Figure 3.12: Basic eWave setup

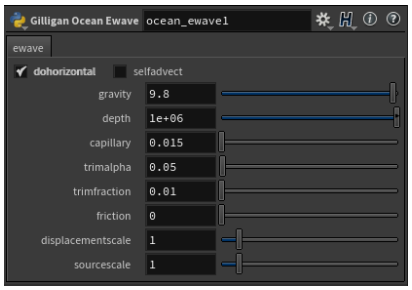

Figure 3.13: Parameter interface of Gilligan Ocean eWave

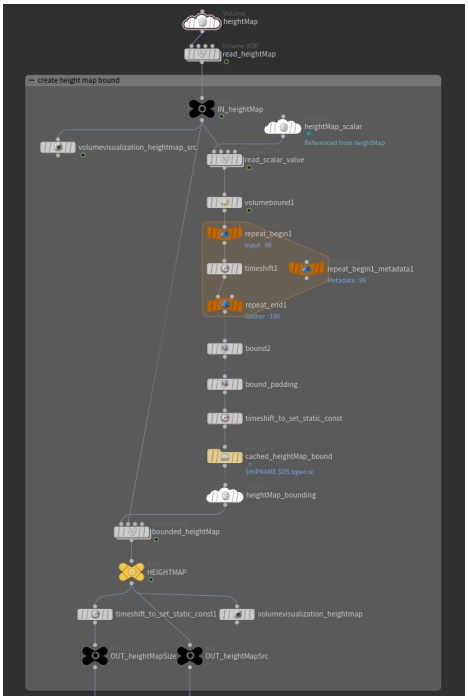

Figure 3.14: Node network of bounding box clipping method for eWave simulation

"Gilligan Ocean eWave Source" keeps updating the height map stored in the eWave object frame by frame by retrieving the height map data. However, the eWave area is much smaller than the entire ocean patch in a lot of cases. Therefore, using the entire eWave height map for eWave simulation is a waste. In order to improve the efficiency of the eWave simulation, a bounding box clipping method can be applied to reduce unnecessary area. This method generates a lower bound of the intersection area among the entire simulation frame range, which is the smallest area of interest for eWave simulation. Then it applies user-defined bound padding to the lower bound, aiming to offer more space for simulation (see Algorithm 2). Houdini Volume Bound node is used to compute the bound of voxels of intersection area in height map volume at each frame. VDB is also a useful structure for reducing voxel usage. Every voxel in VDB stays in two discrete states, namely active or inactive[10]. In scalar density grids of VDB in Houdini, inactive voxels have a default background value (e.g. zero), and active voxels have a value different from this default value. It means that the bounding box of a scalar VDB grid automatically bounds to active voxels for values of our interests. VDB is not used in this system since all of volume structures are Houdini Volume. Figure 3.14 shows the node network of bounding box clipping for eWave height map. The For Loop block merges the

bounding box at each frame. The Timeshift node (the yellow node) keeps the entire node chain connected to "Gilligan Ocean eWave" statically, running only once after setup. "Gilligan Ocean eWave" uses the new LLC, resolution and patch size from bounded height map volume to initialize Gilligan eWave object.

Algorithm 2 Bounding box clipping algorithm

```
1: function BBoxClipping
2:     k = start frame
3:     n = end frame
4:     B_{k-1} = empty BBox
5:     for frame i = k to n do
6:         H_i ← height map volume at frame i
7:         Box_i ← bouding box of voxels of intersection in H_i
8:         B_i = mergeBBox(Box_i, Box_{i-1})
9:     end for
10:    Add bound padding to B_n to generate BBox B
11:    return B
12: end function
```

3.2.2.2 Combination of eWave and Ocean Surface

eWave can be merged with the ocean surface. Gilligan provides internal method to combine the ocean surface and the eWave [17]. This method is only recently available for the use in productions. It is re-implemented in Houdini VOP network to make the entire workflow flow well. The combination of eWave layer and ocean surface layer is much more complicated, which is not the same as merging multiple layers of ocean surfaces by simply adding displacement data of each layer together. This is because that the eWave heightMap is generated after the ocean surface simulation applying to the base ocean geometry. The ocean surface simulation causes distortion of base ocean geometry along vertical direction and horizontal direction, generating wave shapes. However, the eWave simulation happens based on its "local, rectangular coordinates" on the eWave simulation patch, causing shifts from horizontally distorted ocean surface. As it shows in Figure 3.15, the light grey grid inside the red rectangle demonstrates the horizontally distorted ocean surface, and the dark grey rectangle represents the eWave simulation patch. Simply adding eWave displacement onto ocean surface displacement can cause a location shift of the eWave simulation. As the left image in Figure 3.16 shows, the eWaves are not aligned with the interaction object. Therefore, there should be a remap operation for eWave simulation onto the original location of the ocean simulation: the input location should shift along the displacement of ocean surface, as Equation 3.2

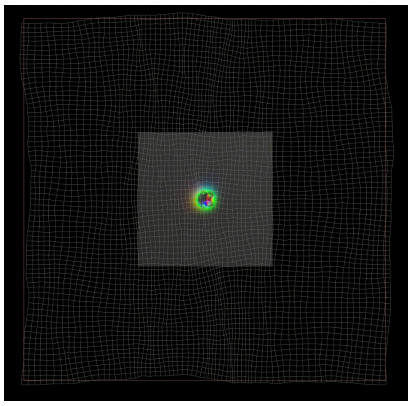

Figure 3.15: Top view of horizontally distorted ocean surface with eWave simulation patch

Figure 3.16: eWave combination without (left) and with (right) remapping

shows. Then the remapped eWaves can be the additional term onto the ocean surface displacement for the combination, showed in Equation 3.3. With remapping, the mis-alignment issue of eWaves is easily fixed. The right image in Figure 3.16 is a good example. This process can be performed in a Houdini Volume VOP node (Figure 3.17), or at the render-time in the ocean shader. Additional scale factor can be applied to the eWave displacement after simulation, providing more controls on the final ocean look.

$$\begin{aligned} \boldsymbol{D_{output}}(x,y) &= \boldsymbol{D_{eWave}}(x',y') \\ x' &= x + \boldsymbol{D_{surface}}(x,y).x \\ y' &= y + \boldsymbol{D_{surface}}(x,y).y \end{aligned} \tag{3.2}$$

$$\boldsymbol{D}(x,y) = \boldsymbol{D_{output}}(x,y) + \boldsymbol{D_{surface}}(x,y) \tag{3.3}$$

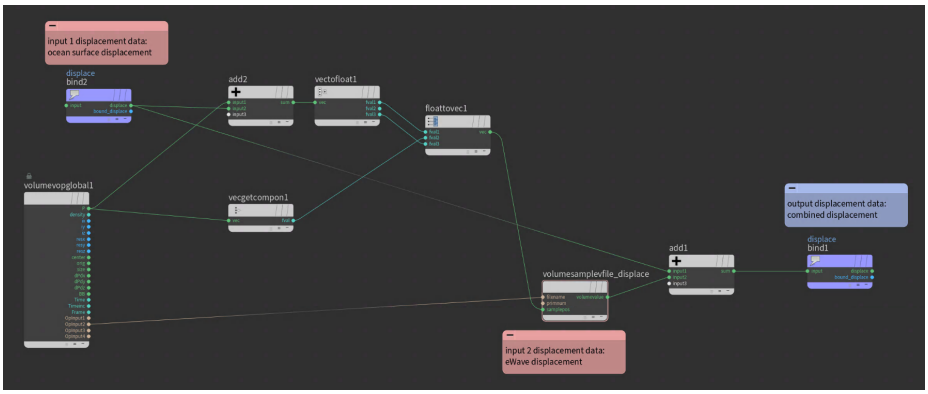

Figure 3.17: Top view of horizontally distorted ocean surface with eWave simulation patch

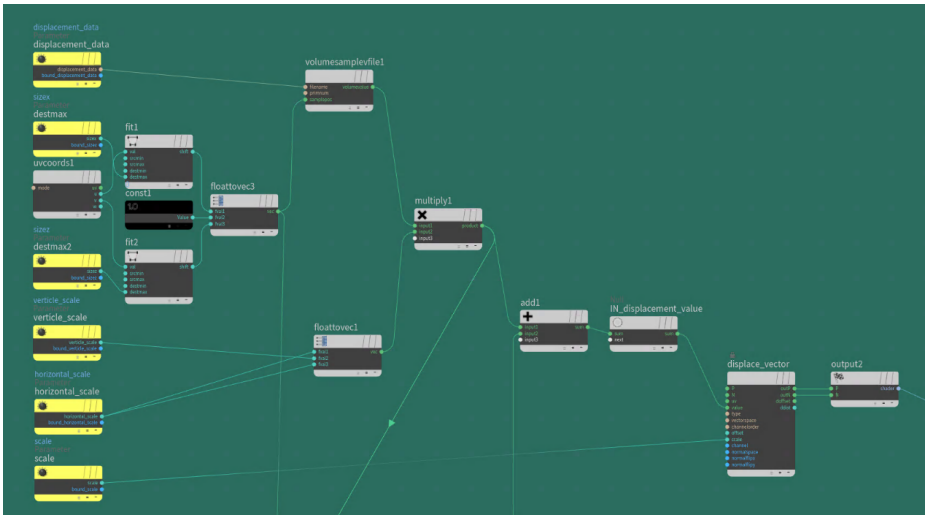

Figure 3.18: Ocean displacement shader

3.2.3 Rendering

Materials in Houdini Mantra encapsulate a surface shader, a displacement shader, and render properties. The node chains representing the surface and displacement shaders feed into Output nodes. My ocean material takes the displacement data from ocean simulation and applies a real-world ocean look onto the base ocean geometry. For my ocean material, surface shader is built with Classic Shader Core node with physically correct parameters of water. Artistic modifications are also allowed, such as changing diffuse color. Displacement shader is built with Displace node using vector displacement type in uv tangent space. Figure 3.18 shows the basic displacement shader network. It uses Houdini Volume Sample Vector node to sample the displacement data from user-

specified displacement file based on geometry's uv coordinates and ocean patch size, allows the user to modify the scale of ocean waves, and feeds the displacement value to the Displace node. As it is mentioned above in section 3.2.2.2, the combination of eWave and ocean surfaces could also be done in material network. If so, the same VOP network is applied to the material network.

There are several parameters to mainly control render quality in this case. First of all, the ocean look is largely dependent on vector displacement. The geometry polygons should be rendered as subdivision. The shading quality under dicing tab is used to control geometric subdivision resolution. Also, the ocean look is mainly contributed from reflection and refraction. The reflect limit and refract limit is the number of times a ray can be reflected or refracted in the scene. These should have a large enough number, otherwise Mantra will render a black background or direct lighting color once the maximum number of reflections or refractions is exceeded[1]

[1]The behaviour is controlled by "At Ray Limit" parameter in Mantra Render node.

Chapter 4

Applications

The Gilligan-Maya workflow has been applied to the production of short animation *Bait*. The Gilligan-Houdini workflow is also capable of mimicking an ocean environment, and it helps to generate other scenarios, such as creating physically plausible wakes or ripples effects and creating magic ocean character effects.

4.1 Ocean Environment Created by Gilligan-Maya Workflow

Gilligan-Maya workflow has been applied to the ocean environment creation in the production of *Bait*. The ocean waves correspond to a clear sunny day on the water. Several floating objects move along the ocean waves. There are two types of ocean environment shots: the ocean surface shot and the underwater shot. They followed the same ocean simulation workflow.

An ocean surface was carefully simulated to match the environment settings. The deep ocean, which is suitable for rounded waves, was selected to simulate the ocean surface waves. Two layers of ocean surfaces in different scale were merged together. Meanwhile, eWaves were simulated to create interaction of the floating objects with the ocean surface. The floating objects were divided into three groups according to the camera layout, with suitable eWave patches specified. Different eWave simulations were applied to these three groups. The ocean surface displacement textures and eWave displacement textures were combined together in Arnold shader in Maya with a control on the scales of each component. A plane with the ocean shader was rendered as the final ocean look shown in Figure 4.1.

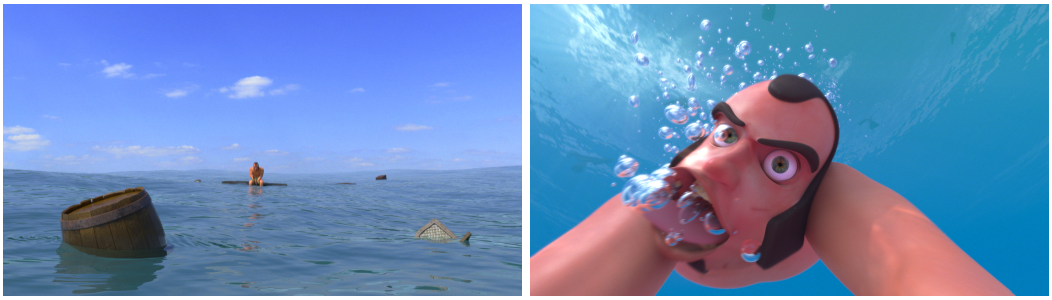

Figure 4.1: Ocean environment created by the ocean simulation following Gilligan-Maya workflow in *Bait(2017)*, Clemson DPA

Figure 4.2: Reference sea environment images shot at Tomales point, CA

4.2 Ocean Environment Created by Gilligan-Houdini Workflow

Ocean environment can also be easily created in Houdini with the Gilligan-Houdini workflow. This section demonstrates an example to create a quiet sea environment. Figure 4.2 shows the reference images I shot at Tomales point, CA, which are quiet deep oceans in clear sunny day with a thin layer of fog beyond the ocean surfaces.

The entire setup for ocean environment includes three parts: ocean surface, ground plane and lighting rig. For the ocean surface, a plane is used as base geometry where ocean surface simulation applies. The ocean surface simulation is built by three layers of ocean surfaces with different patch sizes. The ground plane and the lighting rig are used to determine the color for the ocean surface. The ground plane contributes to the refraction color of the ocean. It serves to mimic the bouncing lights from the ocean floor, which usually has the color of green or blue. In my case, the ground plane has a noisy color of several greenish blue colors. The lighting rig, serves as environment lights, contributes to the reflection color of the ocean. In this case, the lighting rig

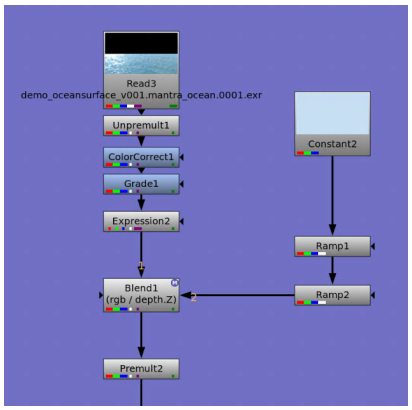

Figure 4.3: Nuke tree to create ocean fog effects

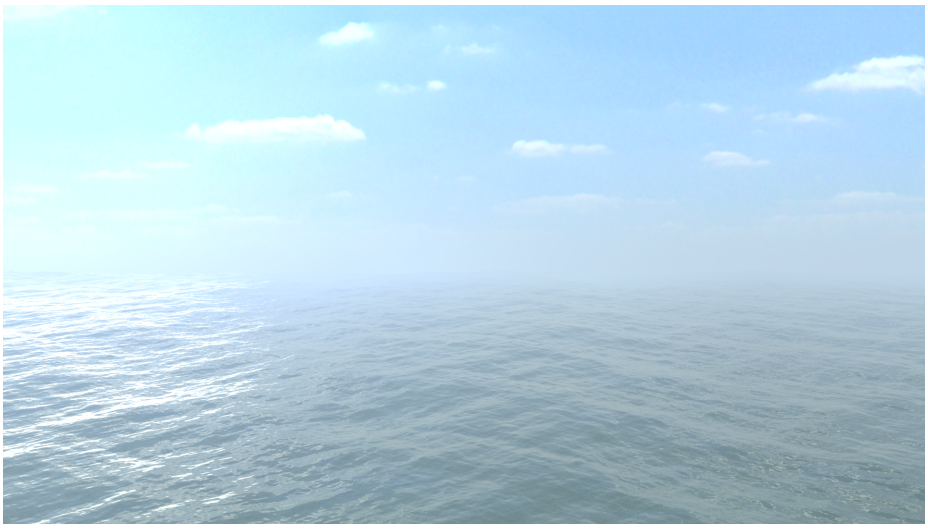

Figure 4.4: CG sea environment created by Gilligan-Houdini

contains an environment light with HDRI image as environment map, and a sun light which create the most specular part of the ocean.

As the reference images shows, there is a thin layer of fog beyond the ocean surface. This can be done via Houdini Atmosphere node with specified fog shader node, which can create a fog effect when rendered. The fog shader computes fog by marching a ray from the eye to the surface being shaded. Several types of fog shader can be used, for example, VEX Lit Fog node can apply a noise to the fog created. However, it costs heavily to use fog shader because the illumination in the scene is computed at each step along the ray. The cheaper way to perform fog effect in this case is to add "fog" color based on Z-depth during composite stage. As it shows in Figure 4.3, a Blend

node is used to mix the ocean and fog color together based on Z-depth value. The final result image shows in Figure 4.4.

4.3 Wakes of Floating Objects

It is a common scenario to place floating objects onto a ocean surface, such as boats and debris, especially for the background scene with a lot of floating objects in the sea environment. There are two key points in this scenario: animating floating objects and generating physically plausible wakes or ripples for these objects.

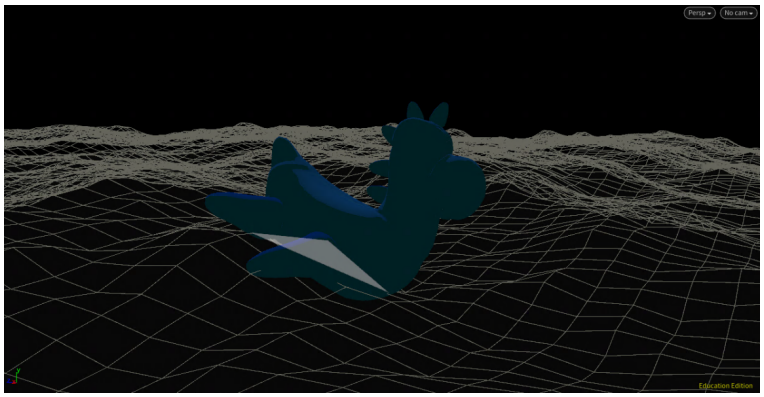

Figure 4.5: An animated floating object (blue object) with proxy geometry (grey triangle) and rough ocean surface

Detailed animation for the secondary floating objects is not necessary. The easiest way to create reasonable animation is to make the floating objects move along with the ocean surface. For example, in the production of *Bait*, an ocean surface geometry generated from Gilligan is used as the reference for animation, as mentioned in section 3.1. Secondary floating objects are attached to the geometry mesh in Maya with some manual modifications. However, this method of attaching to the geometry (the attaching method) only considers the translation of the objects. In real case, the floating objects should move along with ocean surface with nice rotation. It means that a lot of manual animations are required for the attaching method. Therefore, it is worthwhile to find a method to automatically generate reasonable animation for these floating objects. The animation generation module in my toolkit is a useful tool to automatically generate behaviours of floating objects with simple guided animation. This module allows the user to set up a triangle as a proxy geometry to represent the floating object and to indicate the depth of interaction. It lets the user

define a curve as the path of the floating object with moving direction, as well as an animation curve to indicate the relative speed during each frame along the entire time slide, and an ocean surface. With these inputs, this module remaps the animation of the proxy geometry onto the path with correct facing directions at each frame. Then it projects the proxy geometry onto the ocean surface, storing the transformation. The ocean surface is roughly displaced by the ocean surface displacement data, similar with the ocean surface visualization. Finally it applies the transformation to the floating object, generating the animation along the path with movement along the ocean surface (Figure 4.5). The Houdini setup of this module is shown in appendix B.

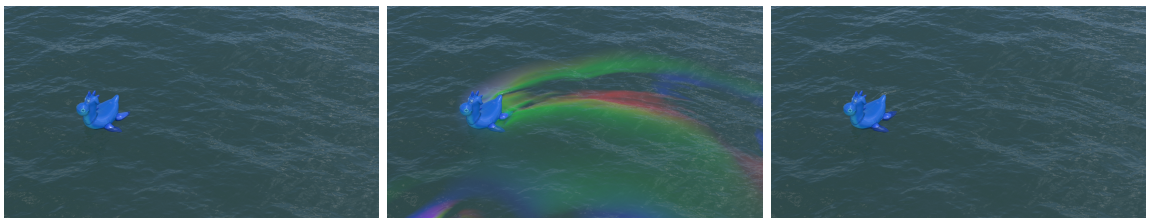

Figure 4.6: eWaves applied onto the ocean surface; the ocean surface without eWaves (left), the ocean surface with eWaves marked as eWave displacement value (middle), the ocean surface with eWaves (right)

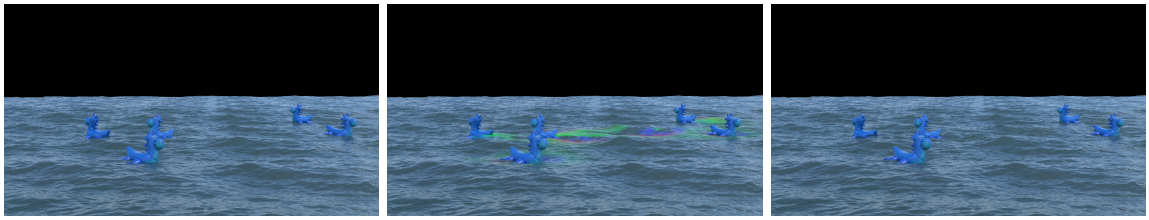

Figure 4.7: Ocean scene with floating objects moved by animation generator; the ocean surface without eWaves (left), the ocean surface with eWaves marked as eWave displacement value (middle), the ocean surface with eWaves (right)

With proxy geometry, ocean surface, moving path and animation curve specified, floating objects can be efficiently animated. Waves and ripples are applied using eWave simulation workflow based on the animated geometry, as introduced in section 3.2.2 (Figure 4.6).

Figure 4.7 presents an example of an ocean scene with floating objects and their wakes. The floating objects are automatically animated by the animation generator module with the guided paths shown in Figure 4.8. Figure 4.9 shows the final composited image.

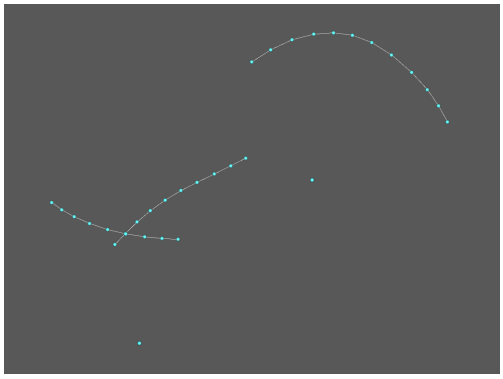

Figure 4.8: Guided path curves of floating objects

Figure 4.9: Composited image of the ocean scene with floating objects in Figure 4.7

4.4 Ocean Character

There is a traditional theory that everything in nature is made up of five basic elements: metal, wood, water, fire, and earth. Characters made from water, as one of these five elements, become a valuable component in many strange stories.

A rubber toy character is selected to demonstrate this concept. This character can deform like ocean, and transition from the normal status into the ocean status. Figure 4.10 shows the concept materials for these two status. The normal status material is rubber, while the ocean status

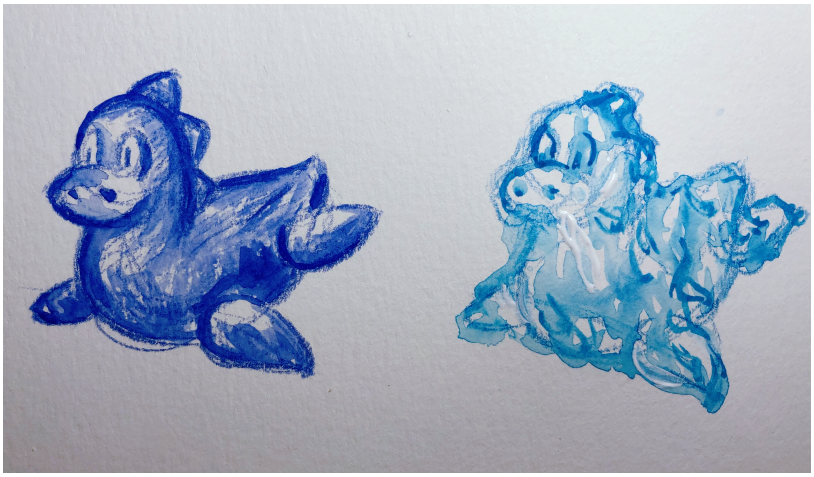

Figure 4.10: Concept materials of the normal status and the ocean status

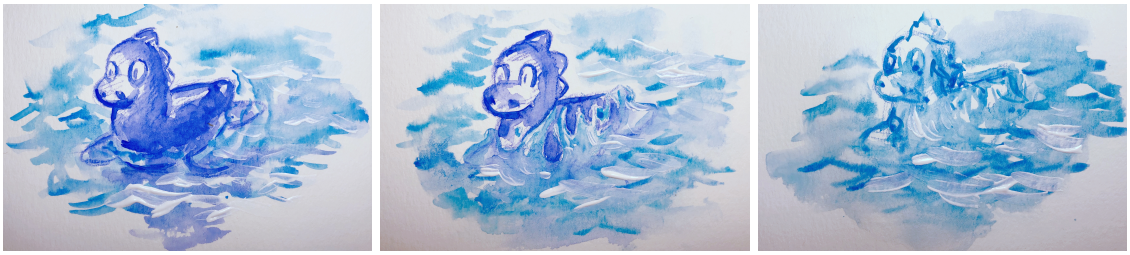

Figure 4.11: Concept images of the transition

material is the ocean with waves generating the water-look deformation. Figure 4.11 shows the transition process of this character, from the normal status turning into ocean status as part of the environment ocean. During the process of the transition, the fluid dynamics are also involved.

The ocean character transition effects contains two simulations: the fluid simulation and the ocean simulation. The fluid simulation, called transition fluid, serves to create the transition process, providing dynamics for the entire transition. The ocean simulation provides the ocean characteristics for the character and the ocean environment for the background. The ocean shader with simulated ocean surface is applied to the character based on a mask of the transition. The transition mask is a point cloud of white and black colors generated from the transition fluid, demonstrating if the area on the character surface has been covered by the transition fluid. In this way, the character transforms from the normal surface to the ocean surface with the transition mask, along with the transition fluid on the top of the surface.

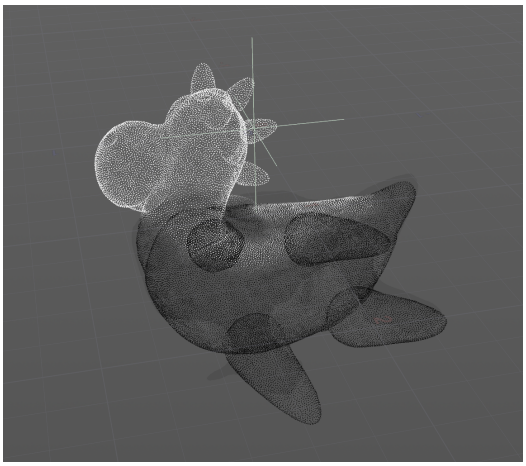

Figure 4.12: An example of transition mask

The transition fluid is created by FLIP simulation with the fluid constrained on the character surface and the movement influenced by a curl noise. The signed distance field (SDF) of the character surface is used to displace the fluid particles onto the character surface using the Equation 4.1.

$$\boldsymbol{P_{new}} = \boldsymbol{P} - \nabla SDF(\boldsymbol{P}) \times SDF(\boldsymbol{P}) \tag{4.1}$$

The transition mask is a point cloud with white (mask information is 1) indicating no transition and black (mask information is 0) representing there is transition; Figure 4.12 shows an example of the transition mask. A Houdini VOP mainly uses XYZ Distance VOP node[1] to generate the mask information at current frame, and combines it with the transition mask at the previous frame, discribed in Equation 4.2.

$$mask_t = min(clamp(xyzdist(\boldsymbol{P}), threshold), mask_{t-1}) \tag{4.2}$$

The character shader is specified by the transition mask. If the transition mask of current position is 1, the normal surface shader is assigned; if the transition mask of current position is 0, the ocean surface shader is assigned. In order to make the ocean part of the character integrate seamlessly with the ocean environment, the underwater part of the ocean character should not be seen through the ocean. This means the underwater part of the ocean character is the water material

[1]The XYZ Distance VOP node finds closest position on a primitive in a given geometry file with an output of the distance value.

Figure 4.13: Three frames of the ocean character transition: the start of the transition, the middle during the transition and the final result of the transition

with water surrounded. Therefore, the inside IOR and the outside IOR for this part both have to be 1.34, the same as the inside IOR of the ocean plane. An underwater mask is calculated during the render-time using Fast Shadow VOP node[2] in the ocean surface shader, and it is used to specify the outside IOR of the ocean character, as 1.34 for underwater parts or 1.00 for parts in the air.

With all the techniques described above, the ocean character transition effects are created, along with an additional eWave layer attached on the ocean plane using the height source map generated from the character and the transition fluid geometry. Figure 4.13 presents three frames during the process of the transition.

[2]Fast Shadow VOP node sends a ray from the position P along the user-specified direction, and it returns 1 if there are no occluding objects found, and 0 0 if the ray hit any object in between.

Chapter 5

Conclusions and Discussion

Gilligan is a useful ocean simulation system to work with the DCC packages. The core technique in Gilligan, spectral waves, are widely used in productions. The Gilligan-Maya workflow has been used to create nice ocean environment for animation production, taking care of effects of the ocean surface and the interaction of objects with the ocean. A more flexible ocean tool with Houdini has been developed based on this workflow. The Houdini-Gilligan workflow contains a Houdini wrapper of Gilligan core simulation glued by Python and several Houdini tools. These workflows implement a relative new technique to combine the eWave layer and the ocean surface layer. Along with Houdini, the artist could have a few of controls for the simulation, while exposed to a lot of accessible simulation data in case of creating advanced effects. It also helps to simplify the production workflows, such as automatically generating floating animation. The ocean simulation tools of Gilligan with Maya and Houdini are capable of creating various ocean related effects for different scenarios, ranging from photo-realistic ocean environments generation, to stylized effects creation, from geometry setup to shader development. What's more, considering the efficency and performace, the toolkit could be further developed by moving the simulation process into the render time by creating custom VEX functions using HDK (Houdini Development Kit) and evaluating VEX functions in Houdini shaders. It could also move from CPU simulation into GPU by using Gilligan's GPU method to improve performance. The supported Houdini tool module could also be improved, for example, the floating animation generator could reduce the animation jitter by adding filter. Meanwhile, considering the artist's requirement, it is also worthwhile to automatically generate initial data for other related simulation, such as fluid simulation.

Appendices

Appendix A Source Code of Houdini Wrapper

This section lists the source code of Gilligan wrapper nodes created by Houdini custom geometry SOP in Python[1].

A.1 Source Code of "Gilligan Ocean Setup" Node

```
node = hou.pwd()

### parms inputs ###
res = node.parmTuple('res').eval()
patchsize = node.parmTuple('size').eval()
llc = node.parmTuple('llc').eval()


### create grid data ###
def grid_generation():
    geo = node.geometry()
    vel_attrib = geo.addAttrib(hou.attribType.Prim, "name", 'noname')
    # bbox
    bbox = hou.BoundingBox(llc[0], -1, llc[1], patchsize[0], 1, patchsize[1])
    # create displacement grids for 3 channels
    grid_r = geo.createVolume(res[0], 1, res[1], bbox)
    grid_g = geo.createVolume(res[0], 1, res[1], bbox)
    grid_b = geo.createVolume(res[0], 1, res[1], bbox)
    # create velocity grids for 3 channels
    grid_vx = geo.createVolume(res[0], 1, res[1], bbox)
    grid_vy = geo.createVolume(res[0], 1, res[1], bbox)
    grid_vz = geo.createVolume(res[0], 1, res[1], bbox)
    # naming
    grid_r.setAttribValue('name', 'displace.x')
    grid_g.setAttribValue('name', 'displace.y')
    grid_b.setAttribValue('name', 'displace.z')
    grid_vx.setAttribValue('name', 'vel.x')
    grid_vy.setAttribValue('name', 'vel.y')
    grid_vz.setAttribValue('name', 'vel.z')
    grids = (grid_r, grid_g, grid_b, grid_vx, grid_vy, grid_vz)
```

[1]The global environment setup and some public functions are located in Houdini Python module source, which is evaluated when the scene file is loaded, and is available to other Python code.

```
31
32      return grids
33
34
35  def cook():
36      grids = grid_generation()
37
38  cook()
```

Listing 1: Python code of “Gilligan Ocean Setup” node

A.2 Source Code of “Gilligan Ocean WaveSurfer” Node

```
1   node = hou.pwd()
2   sim_name = node.name()
3
4   ### parms inputs ###
5   res = node.parmTuple('res').eval()
6   patchsize = node.parmTuple('patchsize').eval()
7
8
9   ### create simulation object ###
10  def simulation_generation():
11      import gilligan.thurston.sim.wavesurfersim as wssim
12
13      wave = wssim.WaveSurferSim(sim_name)
14      # sim parms
15      wave.set('patchsize', '[{sx}, {sy}]'.format(sx=patchsize[0], sy=patchsize[1]))
16      wave.set('patchnxny', '[{nx}, {ny}]'.format(nx=res[0], ny=res[1]))
17      # wave surfer parms
18      wave_surfer_parms = node.parmsInFolder(('wavesurfer', ))
19      for p in wave_surfer_parms:
20          parm_name = p.name()
21          parm_eval = p.eval()
22          print "Set parm {0} to {1}".format(parm_name, parm_eval)
23          wave.set(parm_name, parm_eval)
24
25      print "Create Gilligan wave surfer sim"
26      wave.generate_object()
27
```

```
    return wave


### set wavesurfer sim path ###
def store_path():
    hou.session.store_node_path_attr(node, 'wavesurferpath')


### main ###
def cook():
    wave = simulation_generation()
    # set user data
    node.setCachedUserData('wave', wave)
    # set node path to details attributes
    store_path()

cook()
```

Listing 2: Python code of “Gilligan Ocean WaveSurfer” node

A.3 Source Code of “Gilligan Ocean Merge” Node

```
node = hou.pwd()
inputs = node.inputs()

def read_wavesurfernode(input):
    wavesurfernode = hou.session.node_from_path_attr(input, 'wavesurferpath')
    wave = wavesurfernode.cachedUserData('wave')
    return wave


def store_path():
    hou.session.store_node_path_attr(node, 'wavesurferpath')


### main ###
def cook():
    # merge ocean
    import gilligan.thurston.sim.wavemerge as simmerge
    import gilligan.thurston.sim.wavesurfersim as wssim
```

```
    wave_list = map(lambda x: read_wavesurfernode(x), inputs)

    merged_ocean = simmerge.WaveMerge('base_ocean')
    for wave in wave_list:
        merged_ocean.add_wave(wave)

    merged_ocean.generate_hou_object()
    merged_ocean.verbose = True

    node.setCachedUserData('wave', merged_ocean)
    # store path
    store_path()

cook()
```

Listing 3: Python code of "Gilligan Ocean Merge" node

A.4 Source Code of "Gilligan Ocean eWave" Node

```
node = hou.pwd()
geo = node.geometry()
sim_name = node.name()

# density grid holding height src map data
grid = geo.prims()[0]
resx = grid.resolution()[0]
resz = grid.resolution()[2]

### create simulation object ###
def ewave_generation():
    import gilligan.thurston.sim.ewavesim as ewsim

    ewave = ewsim.eWaveSim(sim_name)

    # ewave setting
    bbox = grid.boundingBox()
    llc = bbox.minvec()
    urc = bbox.maxvec()
    bboxsize = urc - llc
    llc_str = '[{0}, {1}]'.format(llc[0], llc[2])
```

```
22      patchsize_str = '[{0}, {1}]'.format(bboxsize[0], bboxsize[2])
23      patchnxny_str = '[{0}, {1}]'.format(resx, resz)
24
25      ewave.set('llc', llc_str)
26      ewave.set('patchsize', patchsize_str)
27      ewave.set('patchnxny', patchnxny_str)
28
29      # ewave sim parms
30      ewave_parms = node.parmsInFolder(('ewave', ))
31      for p in ewave_parms:
32          parm_name = p.name()
33          parm_eval = p.eval()
34          print "Set parm {0} to {1}".format(parm_name, parm_eval)
35          ewave.set(parm_name, parm_eval)
36
37      ewave.generate_object()
38      print "generate ewave object"
39
40      return ewave
41
42  ### set wavesurfer sim path ###
43  def store_path():
44      geo = node.geometry()
45      # create wavesurfer path detail attribute
46      if geo.findGlobalAttrib('ewavepath') is None:
47          geo.addAttrib(hou.attribType.Global, 'ewavepath', '')
48      geo.setGlobalAttribValue('ewavepath', node.path())
49
50
51  ### cook ###
52  def cook():
53      # generate ewave
54      ewave = ewave_generation()
55      # set user data
56      node.setCachedUserData('ewave', ewave)
57      # set node path to details attributes
58      store_path()
59
```

```
cook()
```

Listing 4: Python code of “Gilligan Ocean eWave” node

A.5 Source Code of “Gilligan Ocean eWaveSrc” Node

```
node = hou.pwd()
geo = node.geometry()

# inputs: 0-ewave object, 1-height src
inputs = node.inputs()
ewavenode = inputs[0]
heightsrcnode = inputs[1]


### convert2image ###
def convert2img(sim_name, grid):
    resx = grid.resolution()[0]
    resz = grid.resolution()[2]

    import gilligan.thurston.camera as cam
    # height src image
    image = cam.Image(sim_name + '_height_src')
    image.set('width', resx)
    image.set('height', resz)
    image.set('channels', 1)
    image.generate_object()

    # convert height src data to gilligan image
    for j in range(0, resz):
        for i in range(0, resx):
            value = grid.voxel((i, 0, j))
            channels = [value]
            cam.set_pixel(image, i, j, channels)

    # set image float array for ewave sim
    image.set_float_array(0)

    return image

```

```
35
36 ### set src for ewave sim ###
37 def set_src(ewave, src):
38     ewave.set_height_source(src)
39
40
41 ### set ewave sim path ###
42 def store_path():
43     geo = node.geometry()
44     # create wavesurfer path detail attribute
45     if geo.findGlobalAttrib('ewavepath') is None:
46         geo.addAttrib(hou.attribType.Global, 'ewavepath', '')
47     geo.setGlobalAttribValue('ewavepath', node.path())
48
49
50 ### cook
51 def cook():
52     ewave = ewavenode.cachedUserData('ewave')
53     # density grid holding height src map data
54     if heightsrcnode is not None:
55         grid = heightsrcnode.geometry().prims()[0]
56         sim_name = ewave.get_name()
57         heightsrc = convert2img(sim_name, grid)
58         set_src(ewave, heightsrc)
59     # set user data
60     node.setCachedUserData('ewave', ewave)
61     # set node path to details attributes
62     store_path()
63
64 cook()
```

Listing 5: Python code of “Gilligan Ocean eWaveSrc” node

A.6 Source Code of “Gilligan Ocean Sim” Node

```
1 node = hou.pwd()
2 geo = node.geometry()
3
4 # inputs: inputs[0]-oceangrid, inputs[1]-ewave, inputs[2]-wavesurfer
5 inputs = node.inputs()
```

```
grid_r = geo.prims()[0]
grid_g = geo.prims()[1]
grid_b = geo.prims()[2]

grid_vx = geo.prims()[3]
grid_vy = geo.prims()[4]
grid_vz = geo.prims()[5]

# simulation config
timestep = node.evalParm('timestep')

sim_wave = None


# update wavesurfersim
def update_wavesurfersim(wavesurfernode):
    wave = wavesurfernode.cachedUserData('wave')
    # update sim
    import gilligan.thurston.sim.wavesurfersim as wssim
    wave.update(timestep)
    print "Update wavesurfersim."
    return wave


# update ewavesim
def update_ewavesim(ewavenode):
    ewave = ewavenode.cachedUserData('ewave')
    import gilligan.thurston.sim.ewavesim as ewsim
    ewave.hou_update(timestep)
    print "Update ewavesim."
    return ewave


def cook():
    wavenode = None
    sim_id = 0
    # sim_handler: {<sim_id>: <sim_update_func>}
    sim_handler = {0: lambda x: None, 1: update_wavesurfersim, 2: update_ewavesim}
    # get input: do wavesurfer sim firstly, if no wavesurfer input, do ewave sim
```

```
    wavenode = hou.session.node_from_path_attr(inputs[2], 'wavesurferpath')
    if wavenode is not None:
        sim_id = 1
    else:
        wavenode = hou.session.node_from_path_attr(inputs[1], 'ewavepath')
        if wavenode is not None:
            sim_id = 2

    ### update sim ###
    sim_wave = sim_handler[sim_id](wavenode)

    ### sim results ###
    # store sim data
    if sim_wave is not None:
        print "Store sim data."
        res = grid_r.resolution()
        resx = res[0]
        resz = res[2]
        for x in range(resx):
            for z in range(resz):
                index = (x, 0, z)
                worldpos = grid_r.indexToPos(index)

                (r, g, b) = sim_wave.displacement_at_world(worldpos[0], worldpos[2])

                grid_r.setVoxel(index, r)
                grid_g.setVoxel(index, g)
                grid_b.setVoxel(index, b)

                grid_vx.setVoxel(index, vx)
                grid_vy.setVoxel(index, vy)
                grid_vz.setVoxel(index, vz)


cook()
```

Listing 6: Python code of “Gilligan Ocean Sim” node

Appendix B Houdini Setup of Floating Animation Generator

Figure 1 shows the entire workflow of floating animation generator module as introduced in section 4.3. The floating object geometry is transformed by transformation calculated from the proxy geometry. The tool calculates the transformation from the user-specified path and ocean surface with defined animation curve. Figure 2 shows the workflow to calculate transformation of making proxy geometry attached to the ocean surface, which is applied back to the floating object geometry. The transformation is calculated from proxy geometry primitive[2], and can be applied onto each point of target geometry, or can be extracted as translation, rotation and scale and be applied on the target geometry by Transform node. The transformations involved in these workflows are easily calculated from the current primitive position/direction and the rest position/direction[3]. Figure 3 is a typical example of transformation calculation. The rotation matrix is calculated by Align VOP node from primitive directions. It multiplies the translation matrix generated from position translation vector and previous transform matrix, calculating the entire transformation.

[2]The proxy geometry has only one primitive.
[3]Normal vector is one of good indications of primitive direction in this case.

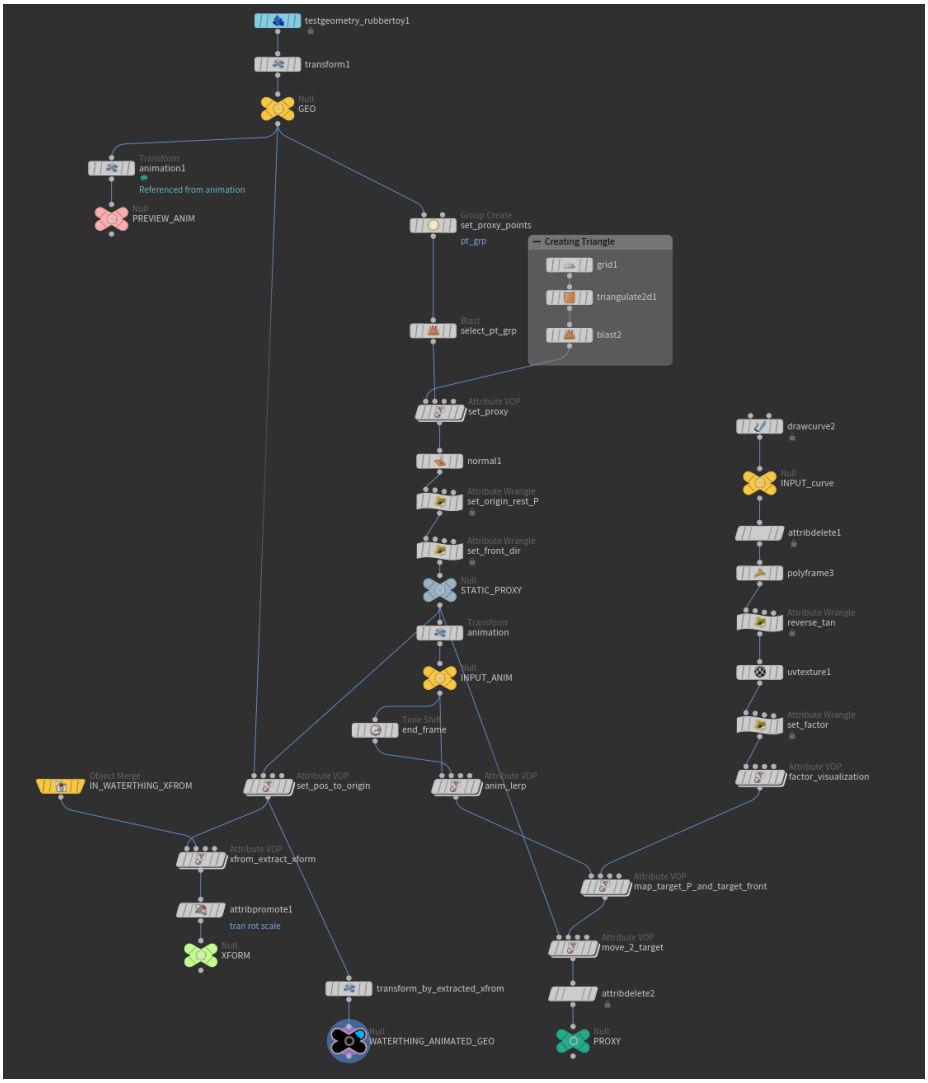

Figure 1: Houdini setup of floating animation generator

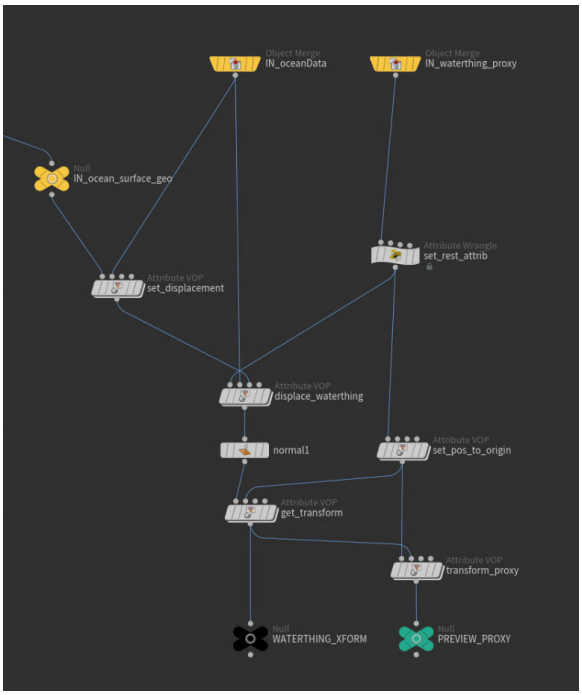

Figure 2: Houdini setup of transformation calculation of projected proxy geometry onto ocean surface

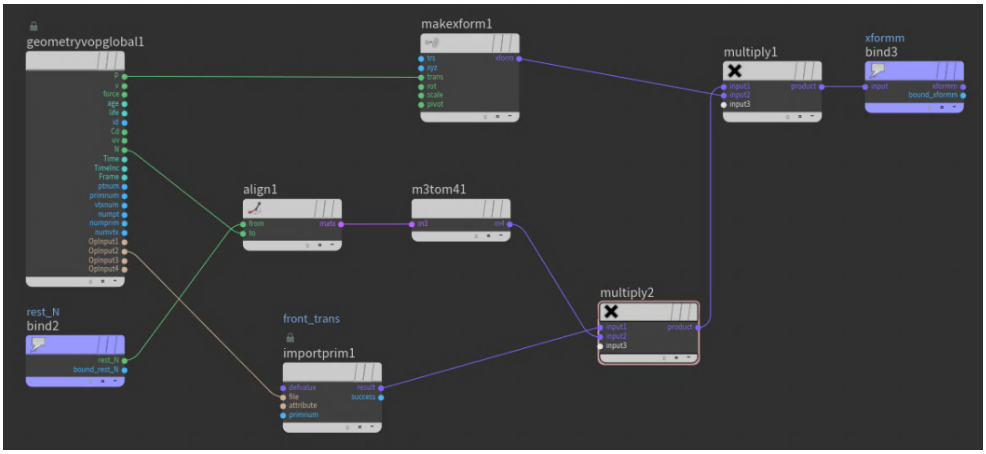

Figure 3: Houdini setup example of transformation calculation VOP

Bibliography

[1] Ocean. http://disney.wikia.com/wiki/Ocean.

[2] AUTODESK. Boss bifröst ocean simulation system. https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/Maya-Bifrost/files/GUID-1D4CE594-25ED-4442-9C22-759E3111EFD7-htm.html.

[3] Emmanuelle Darles, Benoît Crespin, Djamchid Ghazanfarpour, and Jean-Christophe Gonzato. A survey of ocean simulation and rendering techniques in computer graphics. In *Computer Graphics Forum*, volume 30, pages 43–60. Wiley Online Library, 2011.

[4] Ben Frost, Alexey Stomakhin, and Hiroaki Narita. Moana: performing water. In *ACM SIGGRAPH 2017 Talks*, page 30. ACM, 2017.

[5] Jonathan Garcia, Sara Drakeley, Sean Palmer, Erin Ramos, David Hutchins, Ralf Habel, and Alexey Stomakhin. Rigging the oceans of disney's moana. In *SIGGRAPH ASIA 2016 Technical Briefs*, page 30. ACM, 2016.

[6] Kara Lauren Gundersen. *Ocean surface shader*. PhD thesis, Clemson University, 2015.

[7] Rob Hopper and Kai Wolter. The water effects of pirates of the caribbean: Dead men tell no tales. In *ACM SIGGRAPH 2017 Talks*, page 31. ACM, 2017.

[8] Christopher J Horvath. Empirical directional wave spectra for computer graphics. In *Proceedings of the 2015 Symposium on Digital Production*, pages 29–39. ACM, 2015.

[9] Stanislaw R Massel. *Ocean surface waves: their physics and prediction*, volume 36. World scientific, 2013.

[10] Ken Museth. Vdb: High-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics (TOG)*, 32(3):27, 2013.

[11] Sean Palmer, Jonathan Garcia, Sara Drakeley, Patrick Kelly, and Ralf Habel. The ocean and water pipeline of disney's moana. In *ACM SIGGRAPH 2017 Talks*, page 29. ACM, 2017.

[12] SideFX. H15 masterclass: Mantra rendering and texture baking. https://www.sidefx.com/tutorials/h15-masterclass-mantra-rendering-and-texture-baking/, February 2016.

[13] SideFX. Houdini 16 ocean tools. https://www.sidefx.com/tutorials/houdini-16-ocean-tools/, February 2017.

[14] SideFX. What was new in houdini 16. http://www.sidefx.com/docs/houdini/news/16/index.html, 2017.

[15] Walt Disney Animation Studios. Moana. https://www.sidefx.com/community/walt-disney-animation-studios-moana/, 2017.

[16] Moana Technical Team. https://www.instagram.com/disneyanimation/, April 2017.

[17] Jerry Tessendorf. Gilligan: A prototype framework for simulating and rendering maritime environments. https://people.cs.clemson.edu/~jtessen/papers_files/simdoc.pdf, 2017.

[18] Jerry Tessendorf et al. Simulating ocean water. *Simulating nature: realistic and interactive techniques. SIGGRAPH*, 1(2):5, 2001.